
Iványi Antal
PÁRHUZAMOS
ALGORITMUSOK
ELTE Eötvös Kiadó

Iványi Antal

PÁRHUZAMOS ALGORITMUSOK

ELTE Eötvös Kiadó, Budapest, 2005

A könyv az Oktatási Minisztérium támogatásával, a Felsőoktatási Tankönyv- és Szakkönyvtámogatási Pályázat keretében jelent meg.

A könyvet az ELTE IK a pályázat előírásainak megfelelően a http://elek.inf.elte.hu/ címen lévő Elektronikus Könyvtárában közzétette.

Lektorok: Fóthi Ákos, Horváth Zoltán, Kása Zoltán, Pécsy Gábor, Szűcs László

ISBN: 963 463 759 0

Kiadja az
ELTE Eötvös Kiadó
1051 Budapest, Szerb utca 1.
Telefon: 266-9206, Fax: 485-52-26

Honlap: http://www.elte.hu/szervezet/eotvos_kiado.html
Elektronikus cím: eotvoskiado@ludens.elte.hu

Felelős kiadó: Pándi András

Tartalomjegyzék

Előszó **13**

1. Bevezetés **15**

1.1. Alapfogalmak 19

1.2. Hatékonysági mértékek 22

1.3. Pszeudokód 27

1.4. Számítási modellek 31

1.4.1. Számítógépek 31

1.4.2. Párhuzamos gépek 31

1.4.3. Hálózatok 37

1.5. Rekurzió 45

1.6. Véletlenített algoritmusok (⋆) 50

1.6.1. Egyenlőtlenségek 51

1.6.2. Példák 52

1.7. Alsó korlátok 54

1.7.1. Egyszerű számolás 55

1.7.2. Leszámlálás 57

1.7.3. Döntési fák 58

1.7.4. Tanácsadói érvelés 59

1.7.5. Információelméleti érvelés 60
1.7.6. Gráfelméleti érvelés 60
1.8. Anomália 60
1.8.1. Lapcsere 61
1.8.2. Ütemezés 63
1.8.3. Párhuzamos feldolgozás átfedéses memóriával 64
1.8.4. Az anomália elkerülése 65
1.8.5. Párhuzamos korlátozás és szétválasztás 66
1.9. Gyakorlatok 68
1.10. Feladatok 69

2. Párhuzamos gépek 73
2.1. Alapvető módszerek 73
2.1.1. Prefixszámítás 73
2.1.1.1. Prefixszámítás CREW PRAM modellen 75
2.1.1.2. Prefixszámítás EREW PRAM modellen 76
2.1.1.3. Prefixszámítás munkahatékonyan 77
2.1.2. Tömb elemeinek rangsorolása 79
2.1.2.1. Determinisztikus tömbrangsorolás 81
2.1.2.2. Véletlenített listarangsorolás ($\star$) 84
2.2. Kiválasztás 87
2.2.1. Kiválasztás n^2 processzoron 88
2.2.2. Kiválasztás p processzoron 89
2.2.3. Kiválasztás egész számok között 91
2.2.4. Az általános kiválasztási feladat megoldása $n^2/\lg n$ processzoron 93
2.2.5. Munkahatékony véletlenített algoritmus ($\star$) 94
2.3. Összefésülés 96

2.3.1. Logaritmikus idejű algoritmus 97
2.3.2. Páratlan-páros összefésülő algoritmus 97
2.3.3. Munkahatékony algoritmus 101
2.3.4. Egy $O(\lg\lg m)$ idejű algoritmus 103
2.4. Rendezés 104
2.4.1. Páratlan-páros algoritmus 106
2.4.2. Egy véletlenített algoritmus (⋆) 107
2.4.3. Preparata algoritmusa 108
2.4.4. Reischuk véletlenített algoritmusa (⋆) 110
2.5. Gráfalgoritmusok 112
2.5.1. Minmátrix 112
2.5.2. Tranzitív lezárt 114
2.5.3. Összefüggő komponensek 114
2.5.4. Minimális feszítőfa 115
2.6. Gyakorlatok 115
2.7. Feladatok 116

3. Rácsok 119
3.1. Számítási modellek 119
3.2. Csomagirányítás 122
3.2.1. Csomagirányítás láncon 124
3.2.2. Egy mohó algoritmus a PPR megoldására rácson 129
3.2.3. Egy kis várakozási sort használó véletlenített algoritmus (⋆) 131
3.3. Alapfeladatok 132
3.3.1. Üzenetszórás 132
3.3.2. Prefixszámítás 133
3.3.2.1. Prefixszámítás láncon 134

3.3.3. Adatkoncentráció 137
3.3.4. Ritka rendezés 137
3.4. Kiválasztás 139
3.4.1. Véletlenített algoritmus az $p = n$ esetre (⋆) 140
3.4.2. Véletlenített algoritmus a $p < n$ esetre (⋆) 140
3.4.3. Determinisztikus algoritmus a $p < n$ esetre 141
3.5. Összefésülés 144
3.5.1. Rangon alapuló összefésülés láncon 144
3.5.2. Páratlan-páros összefésülés láncon 145
3.5.3. Páratlan-páros összefésülés négyzeten 146
3.6. Rendezés 148
3.6.1. Rendezés láncon 148
3.6.1.1. Rangsoroló rendezés láncon 148
3.6.1.2. Páratlan-páros felcserélő rendezés láncon 148
3.6.1.3. Páratlan-páros összefésülő rendezés láncon 149
3.6.2. Rendezés négyzeten 150
3.6.2.1. Schearson rendező algoritmusa 150
3.6.2.2. Páratlan-páros összefésülő rendezés 150
3.7. Gráfalgoritmusok 151
3.7.1. Kocka 152
3.7.1.1. Minmátrix számítása 152
3.7.1.2. Irányított gráf tranzitív lezártja 153
3.7.1.3. Összefüggő komponensek meghatározása 153
3.7.2. Négyzet 153
3.7.2.1. Tranzitív lezárt 154
3.7.2.2. Legrövidebb utak 155
3.7.2.3. Konvex burok 155
3.8. Gyakorlatok 158

3.9. Feladatok 159

4. Hiperkocka 163

4.1. Számítási modellek 163
4.1.1. Hiperkocka 163
4.1.2. Pillangó hálózat 165
4.1.3. Hálózatok beágyazása 167
4.1.3.1. Gyűrű beágyazása 168
4.1.3.2. Tórusz beágyazása 170
4.1.3.3. Bináris fa beágyazása 171
4.2. Csomagirányítás 173
4.2.1. Mohó algoritmus 173
4.2.2. Véletlenített algoritmus 175
4.2.3. Az első fázis elemzése 177
4.2.3.1. A sorméret elemzése 178
4.3. Alapvető algoritmusok 179
4.3.1. Üzenetszórás fában 179
4.3.2. Prefixszámítás fán 180
4.3.3. Adatkoncentráció 183
4.3.4. Kisszámú elem rendezése hiperkockán 183
4.4. Kiválasztás 186
4.4.1. Véletlenített algoritmus a $p = n$ esetre $(*)$ 186
4.4.2. Véletlenített algoritmus a $p < n$ esetre $(*)$ 186
4.4.3. Determinisztikus algoritmus a $p < n$ esetre 188
4.5. Összefésülés 189
4.5.1. Páratlan-páros összefésülés 190
4.5.2. Biton összefésülés 191
4.6. Rendezés 192

4.6.1. Páratlan-páros összefésülő rendezés 192
4.6.2. Biton rendezés 193
4.7. Gráfalgoritmusok 193
4.7.1. Minmátrix meghatározása 193
4.7.2. Tranzitív lezárt 195
4.7.3. Összefüggő komponensek 195
4.7.4. Legrövidebb utak 195
4.7.5. Konvex burok 196
4.8. Gyakorlatok 196
4.9. Feladatok 197

5. Szinkronizált hálózat 199
5.1. Számítási modell 199
5.2. Vezető választása 200
5.2.1. Vezetőválasztás megoldhatatlansága gyűrűben 201
5.2.2. Vezetőválasztás gyűrűben 202
5.2.2.1. LeLann algoritmusa 202
5.2.2.2. Chang és Roberts algoritmusa 205
5.2.2.3. Hirschberg és Sinclair algoritmusa 208
5.2.2.4. I ˝- algoritmus 208
5.2.2.5. Alsó korlát az üzenetszámra 209
5.2.3. Vezetőválasztás fában 209
5.2.4. Vezetőválasztás általános hálózatban 210
5.2.4.1. M - algoritmus 211
5.2.4.2. O - - algoritmus 211
5.2.5. Alsó korlát az üzenetek számára 211
5.3. Megegyezés 212
5.3.1. Megegyezés vonalhibák esetében 212

5.3.2. Megegyezés processzorhibák esetében 213
5.3.3. *k*-megegyezés 213
5.3.4. Közelítő megegyezés 214
5.4. Gyakorlatok 214
5.5. Feladatok 221

6. Hagyományos és elektronikus irodalom 225
6.1. Megjegyzések az 1. fejezethez 225
6.2. Megjegyzések a 2. fejezethez 230
6.3. Megjegyzések a 3. fejezethez 230
6.4. Megjegyzések a 4. fejezethez 231
6.5. Megjegyzések az 5. fejezethez 231

Jelölések 239

Angol szakkifejezések 244

Magyar szakkifejezések 247

Irodalomjegyzék 251

Lelőhelyjegyzék 290

Névmutató 307

Tárgymutató 319

Előszó

A könyv az ELTE programtervező matematikus szakának I/2. sávjában a *Számítógépes rendszerek* tantárgy, valamint az Informatikai Doktori Iskolában a *Párhuzamos és osztott algoritmusok elemzése* tantárgy keretében tartott előadások anyagát tartalmazza.

A könyv hat fejezetből (*Bevezetés, Párhuzamos gép, Rács, Kocka, Szinkronizált hálózat, Hagyományos és elektronikus irodalom*) és hét további részből (*Jelölések, Angol szakkifejezések, Magyar szakkifejezések, Irodalomjegyzék, Lelőhelyjegyzék, Névmutató, Tárgymutató*) áll.

A második fejezetet Horváth Zoltán, a harmadikat Szűcs László, a negyediket Pécsy Gábor, a könyv többi részét Kása Zoltán lektorálta. Fóthi Ákos sokat segített a könyv felépítésének és stílusának kialakításában.

Köszönöm a könyv lektorainak a sok hasznos észrevételt. Lényeges segítséget jelentett, hogy a lektorok nem csak a saját részüket nézték át. Kollégáim közül köszönöm Balogh Ádám (ELTE), Dózsa Gábor (SZTAKI), Fábián Mária (ELTE), Fóthi Ákos (ELTE), Miletics Edit (Széchenyi István Egyetem), Pethő Attila (Debreceni Egyetem), Rét Anna (Műszaki Könyvkiadó), Scharnitzky Viktorné (ELTE), Sima Dezső (BMF), Simon Péter (ELTE) és Szili László (ELTE) javaslatait. A korábbi változatok alapján vizsgázó hallgatók közül pedig Balázs Gábor (ELTE), Baksa Klára (ELTE), Dévai Gergely (ELTE), Hajdu Tamás (ELTE), Hegyessy Tamás (ELTE), Hermann Péter (ELTE), Kapinya Judit (ELTE), Kovács

Péter (ELTE), Metykó Beáta (ELTE) és Szalai Róbert (ELTE) segítettek.

A könyv ábráit Locher Kornél rajzolta. A könyv kézirata a HLATEX kiadványszerkesztővel készült, amelyet az elmúlt években fejlesztettünk ki Belényesi Viktorral. Az irodalomjegyzéket Iványi Anna tette élővé.

A párhuzamos algoritmusok szakirodalma nagyon gyorsan fejlődik. Ezért a könyvet az ELTE IK

http://elek.inf.elte.hu/

címen lévő *Elektronikus Könyvtárában* folyamatosan karbantartjuk – beleértve a *külföldi és hazai hivatkozások* aktuális listáját is (hiperszöveg formájában, így a honlap látogatói kattintással közvetlenül elérhetik az idézett művek szerzőinek honlapját, és olyan elektronikus könyvtárakat, ahonnan az idézett cikkek letölthetők).

Az *egyes szövegrészek aláhúzása* azt jelzi (itt az előszóban, az irodalomjegyzékben és a lelőhelyjegyzékben), hogy a PDF és HTML változatban a címek élnek, a PS változatban pedig kék színnel ki vannak emelve.

Kérem a könyv Olvasóit, hogy észrevételeiket és javaslataikat írják meg a

tony@compalg.elte.hu

címre. Különösen köszönöm a témával kapcsolatos új feladatokat.

Budapest, 2005. április 26.

Iványi Antal

1. Bevezetés

A számítógépes feladatmegoldás természetes és hagyományos világa a *soros adatfeldolgozás*. A különböző alkalmazási területek nagy teljesítményigényének és környezetünk párhuzamos jellegének hatására azonban rohamosan fejlődik a párhuzamos feldolgozás is.

Párhuzamos adatfeldolgozás. Egy feladat párhuzamos megoldása ugyanúgy 3 lépésből áll, mint a soros megoldás. Először meg kell érteni és pontosan *meg kell fogalmazni* a feladatot – ez a lépés hasonló a soros feldolgozás első lépéséhez. Mivel a soros adatfeldolgozás számára ismert feladatokat oldunk meg, a problémák megértése – az eddigi tapasztalatok szerint – az olvasók többsége (például harmadéves programtervező és informatika szakos hallgatók) számára nem jelent gondot.

A második lépésben választunk egy ismert architektúrát (esetleg tervezünk egy újat) és ahhoz *tervezünk egy megoldási algoritmust*. Ez vagy új algoritmus, vagy egy soros algoritmus párhuzamosított változata. Könyvünk tárgya a párhuzamos adatfeldolgozásnak ez a része, azaz a *párhuzamos algoritmusok tervezése és elemzése*.

Végül a harmadik lépésben az algoritmust a meglévő programozási módszerek valamelyikével *párhuzamos program formájában megvalósítjuk* (itt is szóba jön új programozási módszer alkalmazása).

Párhuzamos algoritmusok. Ha egy feladatot együttműködő processzorok segítségével oldunk meg, akkor a számítási idő rendszerint csökken, mivel bizonyos műveletek egyidejűleg elvégezhetők. Ennek a csökkenésnek az ára a nagyobb beruházásigény és az elvégzendő munka mennyiségének növekedése.

A párhuzamos algoritmusokat különböző szempontok szerint szokták osztályozni.

Az egyik szempont a processzorok működésének *időzítése*. Eszerint vannak összehangoltan dolgozó, ún. *szinkronizált processzorok* és egymástól függetlenül dolgozó, ún. *aszinkron processzorok*. Emellett vannak *hibrid* megoldások, ahol a processzorok részben összehangoltan dolgoznak.

Egy másik osztályozási szempont az együttműködő processzorok közötti információcsere módja, azaz a *kommunikációs modell*. Ez az információcsere történhet a közös memória segítségével és/vagy üzenetek küldésével és fogadásával.

Fontos a *rendszer megbízhatóságával* kapcsolatos felfogásunk: hibátlan működést tételezünk fel vagy megengedünk bizonyos típusú hibákat (például a processzorok vagy az adatátviteli vonalak meghibásodását).

Lényeges az az *architektúra*, amelyre algoritmusainkat tervezzük. A második fejezetben röviden áttekintjük a párhuzamos számítógépek főbb típusait, az elemzésekhez azonban a számítási modelleknek nevezett *absztrakt gépeket* használjuk.

A párhuzamos algoritmusok közül csak a szinkronizált modelleken megvalósíthatókat tárgyaljuk. Eközben feltételezzük, hogy a processzorok, adatátviteli vonalak, közös és saját memória – azaz a rendszer minden eleme – megbízhatóan működnek.

Előismeretek. A tárgyalt anyag nagy része az informatikai képzés alapvető kurzusait (adatszerkezetek, algoritmusok, analízis, diszkrét matematika, programozás) sikeresen elvégző hallgatók számára érthető. A véletlenített algoritmusok elemzése a binomiális eloszlás néhány tulajdonságán alapul, ezért ezekben az alfejezetekben a valószínűségszámítási ismeretekre is támaszkodunk. Ezeket a részeket a címekben és a tartalomjegyzékben csillaggal ($*$) jelöltük. A feladatok egy részének megoldásához hasznosak az operációkutatással, optimalizálással és szimulációval kapcsolatos ismeretek.

A fejezetek egymásra épülése. Bár helyenként felhasználunk a könyv korábbi részeiben bemutatott algoritmusokat, a bevezető első fejezet elolvasása után a többi fejezet egymástól függetlenül is érthető. Ha azonban az Olvasót például a konvex burok hiperkockán való meghatározása érdekli, célszerű a párhuzamos gépen és a rácson alkalmazott algoritmusokkal is megismerkedni (a tartalomjegyzék és a tárgymutató segít a *visszalapozásban*).

Tartalom. A könyv hat fejezetből és hét további részből áll.

Az első fejezetben (*Bevezetés*) előkészítjük a további anyagot: megadjuk az alapvető meghatározásokat, ismertetjük a felhasznált számítási modelleket és pszeudokódot, foglalkozunk a rekurzív algoritmusokkal és a rekurzív egyenletekkel, a véletlenített algoritmusokkal, az alsó korlátokkal és az anomáliával.

A további négy fejezetben párhuzamos algoritmusokat ismertetünk és elemzünk – az algoritmusok megvalósítására használt számítási modellek (és az azok alapjául szolgáló architektúrák) szerint csoportosítva: a szinkronizált processzorok kapcsolatát párhuzamos közvetlen hozzáférésű gépek (*Párhuzamos gép*), rácsok (*Rács*), kockák (*Kocka*) és tetszőleges gráfok (*Szinkron hálózat*) segítségével adjuk meg.

A hatodik fejezetben (*Hagyományos és elektronikus irodalom*) a könyv írásához felhasznált és az egyes témák részletesebb megisme- réséhez ajánlott –

nyomtatott és elektronikus formában, magyar és idegen nyelveken hozzáférhető – dokumentumok tartalmát és lelőhelyi adatait foglaljuk össze.

Módszertan. A könyv felépítésével és az alkalmazott tipográfiai eszközökkel igyekeztünk megkönnyíteni az anyag megértését. A szövegben gyakran alkalmaztunk magyarázattal ellátott ábrákat és példákat. Az egyes fejezetek végén gyakorlatok és feladatok vannak. A gyakorlatok a fejezet anyagának jobb megértését segítik elő és az anyag ismeretében rendszerint gyorsan megoldhatók. A feladatok megoldása önálló gondolkodást és esetenként több matematikai ismeretet igényel.

Az algoritmusok elemzését nem törtük meg hivatkozásokkal, viszont a hatodik fejezetben témakörönként összefoglaltuk az elsősorban ajánlott szakkönyvek és cikkek adatait, és alternatív megoldásokra is utaltunk.

A könyv végén összefoglaltuk az alkalmazott jelöléseket (*Jelölések*), és megadtuk az angol szakkifejezések (*Angol szakkifejezések*) magyar, valamint a magyar szakkifejezések (*Magyar szakkifejezések*) angol megfelelőjét. A bizonyítások végét ■, a példák végét ♠ jelzi. A definíciókban az új fogalom nevét *dőlt betűkkel* írtuk. A definíciókra a ♣ jellel is felhívtuk a figyelmet. A sorszámozott képletek tördelésénél az Olvasók kényelmét szolgáló amerikai stílust követtük (azaz minden relációjel új sorba kerül – az ilyen képletek sorfolytonosan olvasandók). Mivel a vesszőnek gyakran van a szövegben nyelvtani szerepe, a számokban tizedespontot használunk.

Az irodalomjegyzékben együtt adjuk meg a hazai és külföldi forrásokat. Az idegen nyelvű szakirodalomból csak a klasszikus és a viszonylag friss műveket soroljuk fel. A magyar nyelvű anyag összeállítása során – az algoritmusok elemzésével foglalkozó műveket tekintve – teljességre törekedtünk. Az elektronikus formában elérhető dokumentumoknál megadtuk a hálózati címet is. Minden dokumentumnál feltüntettük azoknak a szövegrészeknek az azonosítóját, amelyekből hivatkozás van az adott dokumentumra (pl. 2.7. a megfelelő alfejezetre, 113 pedig az irodalomjegyzék 113-as sorszámú elemében lévő hivatkozásra utal).

A névmutató a könyvben előforduló neveket tartalmazza – teljességre törekedve.

A tárgymutatóban *dőlt számok* jelzik az egyes fogalmak definiálásának helyét. Az előfordulási helyek felsorolásánál megelégedtünk a lényegesnek tartott helyekre való utalással.

1.1. Alapfogalmak

A párhuzamos algoritmusok tárgyalása a soros algoritmusokra épül, ezért a szokásos fogalmak mellett a soros algoritmusokkal kapcsolatos definíciókat is megadjuk.

Az algoritmusok elemzése – a helyesség bizonyítása mellett – a végrehajtáshoz szükséges erőforrások (ez számítógépen megvalósított algoritmus esetében lehet processzoridő, memóriakapacitás – számítási modellen futó algoritmus esetében pedig memóriarekesz, kommunikációs üzenet, műveleti lépés) mennyiségének meghatározását is jelenti. Gyakran nem tudjuk vagy nem akarjuk az igényelt erőforrás mennyiségét pontosan megadni. Ilyenkor megelégszünk az igény nagyságrendjének jellemzésével.

Ennek a jellemzésnek a jól bevált eszközei az Ω, O, Θ, o és ω jelölések. Mivel az igények általában nemnegatívak, ezért az alábbi meghatározásokban mindenütt feltesszük, hogy az f és g függvények a pozitív egészek halmazán vannak értelmezve, az $f(n)$ és $g(n)$ függvényértékek pedig nemnegatív valós számok.

Az O jelöléssel aszimptotikus felső, az Ω jelöléssel aszimptotikus alsó korlátot tudunk adni, míg a Θ jelöléssel pontosan meg tudjuk adni a vizsgált függvény aszimptotikus viselkedését.

$O(g(n))$ (ejtsd: *nagy ordó*) ($\clubsuit$) azon $f(n)$ függvények halmazát jelenti, amelyekhez léteznek olyan c pozitív valós és n_0 pozitív egész állandók, hogy

$$\text{ha } n \geq n_0, \quad \text{akkor } f(n) \leq cg(n). \tag{1.1}$$

$\Omega(g(n))$ (ejtsd: *nagy omega*) (♣) azon $f(n)$ függvények halmazát jelenti, amelyekhez léteznek olyan c pozitív valós és n_0 pozitív egész állandók, hogy

$$\text{ha } n \geq n_0, \text{ akkor } f(n) \geq cg(n). \tag{1.2}$$

$\Theta(g(n))$ (ejtsd: *nagy teta*) (♣) azon $f(n)$ függvények halmazát jelenti, amelyekhez léteznek olyan pozitív valós c_1, c_2 és pozitív egész n_0 állandók, hogy

$$\text{ha } n \geq n_0, \text{ akkor } c_1 g(n) \leq f(n) \leq c_2 g(n). \tag{1.3}$$

Az elemzésekben arra törekszünk, hogy Θ típusú becsléseket adjunk, amihez azonos argumentumfüggvényt tartalmazó O és Ω típusú becslésekre van szükség. Ha egy becslésben hangsúlyozni akarjuk, hogy a becslés nem éles, akkor hasznos a o és a ω jelölés.

$o(g(n))$ (ejtsd: *kis ordó*) (♣) azon $f(n)$ függvények halmazát jelenti, melyekre teljesül, hogy minden pozitív valós c állandóhoz megadható egy pozitív egész n_0 úgy, hogy

$$\text{ha } n \geq n_0, \text{ akkor } f(n) \leq cg(n). \tag{1.4}$$

$\omega(g(n))$ (ejtsd: *kis omega*) (♣) azon $f(n)$ függvények halmazát jelenti, melyekre teljesül, hogy minden pozitív valós c állandóhoz megadható egy pozitív egész n_0 úgy, hogy

$$\text{ha } n \geq n_0, \text{ akkor } f(n) \geq cg(n). \tag{1.5}$$

Ha $f(n) = o(g(n))$, akkor

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0, \tag{1.6}$$

és ha $f(n) = \omega(g(n))$, akkor

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty. \tag{1.7}$$

Az 1.1. táblázatban összefoglaltuk a függvények növekedésével kapcsolatban leggyakrabban használt jelöléseket és kifejezéseket. A táblázatban szereplő korlátok tetszőleges erőforrással kapcsolatban használhatók – elemzéseinkben azonban elsősorban lépésszámokra vonatkoznak.

1.1. táblázat. Függvények növekedésének leggyakoribb korlátai

Sorszám	Növekedési korlát képlettel	Korlát típusa szóval
1	$\Theta(1)$	konstans
2	$\Theta(\lg^\star n)$	majdnem konstans
3	$o(\lg n)$	szublogaritmikus
4	$\Theta(\lg n)$	logaritmikus
5	$\Theta(\lg^{\Theta(1)} n)$	polilogaritmikus
6	$\omega(\lg n)$	szuperlogaritmikus
7	$o(n)$	szublineáris
8	$\Theta(n)$	lineáris
9	$\omega(n)$	szuperlineáris
10	$\Theta(n^2)$	négyzetes
11	$\Theta(n^3)$	köbös
12	$o(n^{\Theta(1)})$	szubpolinomiális
13	$\Theta(n^{\Theta(1)})$	polinomiális
14	$\omega(n^{\Theta(1)})$	szuperpolinomiális

A táblázatban a *szub* kifejezést *kisebb*, a *szuper* kifejezést pedig *nagyobb* értelemben használtuk. Érdemes megemlíteni, hogy szokás a *kisebb vagy egyenlő*, illetve a *nagyobb vagy egyenlő* értelmű használat is.

A könyvben a szuperpolinomiális kifejezés szinonimájaként fogjuk használni az *exponenciális* (♣) jelzőt. Ezzel az *exponenciális lépésszámot* (♣) a matematikában szokásosnál tágabban definiáltuk: ha egy algoritmus lépésszáma felülről nem korlátozható polinommal, akkor exponenciálisnak nevezzük.

1.2. Hatékonysági mértékek

Az algoritmusok elemzése során az igényelt erőforrások mennyiségét *abszolút* és *relatív* mennyiségekkel jellemezzük.

Ezeknek a mennyiségeknek a célszerű megválasztása attól is függ, hogy az algoritmus konkrét vagy absztrakt gépen (számítási modellen) fut.

Jelöljük $W(n, \pi, \mathcal{A})$-vel, illetve $W(n, \pi, p, \mathcal{P})$-vel azoknak a lépéseknek a számát, amelyekkel a π problémát az $\mathcal{A}$ soros, illetve a $\mathcal{P}$ párhuzamos algoritmus – (utóbbi p processzort felhasználva) – n méretű feladatokra legrosszabb esetben megoldja.

Hasonlóképpen jelöljük $B(n, \pi, \mathcal{A})$-vel, illetve $B(n, \pi, p, \mathcal{P})$-vel azoknak a lépéseknek a számát, amelyekkel a π problémát az $\mathcal{A}$ soros, illetve a $\mathcal{P}$ párhuzamos algoritmus (utóbbi p processzort felhasználva) – n méretű feladatokra legjobb esetben megoldja.

Jelöljük $N(n, \pi)$-vel, illetve $N(n, \pi, p)$-vel azoknak a lépéseknek a számát, amelyekre az n méretű π feladat megoldásához mindenképpen szüksége van bármely soros, illetve bármely párhuzamos algoritmusnak – utóbbinak akkor, ha legfeljebb p processzort vehet igénybe.

Tegyük fel, hogy minden n-re adott a π feladat n méretű konkrét előfordulásainak $D(\pi, n)$ eloszlásfüggvénye. Ekkor legyen $E(n, \pi, \mathcal{A})$, illetve $E(n, \pi, p, \mathcal{P})$ annak az időnek a várható értéke, amennyi alatt a π problémát n méretű feladatokra megoldja az $\mathcal{A}$ soros, illetve a $\mathcal{P}$ párhuzamos algoritmus – utóbbi p processzort felhasználva.

A tankönyvekben az elemzések során gyakori az a feltételezés, hogy az azonos méretű bemenetek előfordulási valószínűsége azonos. Ilyenkor *átlagos* (♣) lépésszámról beszélünk, amit $A(n, \mathcal{A})$-val jelölünk.

A W, B, N, E és A jellemzők függnek attól a számítási modelltől is, amelyen az algoritmust megvalósítjuk. Az egyszerűség kedvéért feltesszük, az algoritmus egyúttal a számítási modellt is egyértelműen meghatározza.

Amennyiben a szövegkörnyezet alapján egyértelmű, hogy melyik problémáról van szó, akkor a jelölésekből π-t elhagyjuk.

Ezek között a jellemzők között érvényesek az

$$
\begin{aligned}
N(n) &\leq B(n, \mathcal{A}) && (1.8)\\
&\leq E(n, \mathcal{A}) && (1.9)\\
&\leq W(n, \mathcal{A}) && (1.10)
\end{aligned}
$$

egyenlőtlenségek. Hasonlóképpen a párhuzamos algoritmusok jellemző adataira az

$$
\begin{aligned}
N(n, p) &\leq B(n, \mathcal{P}, p) && (1.11)\\
&\leq E(n, \mathcal{P}, p) && (1.12)\\
&\leq W(n, \mathcal{P}, p) && (1.13)
\end{aligned}
$$

egyenlőtlenségek teljesülnek. Az átlagos lépésszámra pedig a

$$
\begin{aligned}
B(n, \mathcal{A}) &\leq A(n, \mathcal{A}) && (1.14)\\
&\leq W(n, \mathcal{A}), && (1.15)
\end{aligned}
$$

illetve

$$
\begin{aligned}
B(n, \mathcal{P}, p) &\leq A(n, \mathcal{P}, p) && (1.16)\\
&\leq W(n, \mathcal{P}, p) && (1.17)
\end{aligned}
$$

áll fenn.

Hangsúlyozzuk, hogy ezek a jelölések nem csak lépésszámra, hanem az algoritmusok más jellemzőire – például memóriaigény, küldött üzenetek száma – is alkalmazhatók.

Ezután relatív jellemzőket, úgynevezett *hatékonysági mértékeket* definiálunk.

A relatív lépésszám azt mutatja, hányszor kisebb a párhuzamos algoritmus lépésszáma a soros algoritmus lépésszámánál.

A $\mathcal{P}$ párhuzamos algoritmusnak az $\mathcal{A}$ soros algoritmusra vonatkozó *relatív lépésszáma* (♣)

$$g(n, \mathcal{A}, \mathcal{P}) = \frac{W(n, \mathcal{A})}{W(n, p, \mathcal{P})} \ . \tag{1.18}$$

Ha a relatív lépésszám egyenesen arányos a felhasznált processzorok számával, akkor lineáris relatív lépésszámról beszélünk. Ha a $\mathcal{P}$ párhuzamos és az $\mathcal{A}$ soros algoritmusra

$$\frac{W(n, \mathcal{A})}{W(n, p, \mathcal{P})} = \Theta(p), \tag{1.19}$$

akkor $\mathcal{P}$ A-ra vonatkozó relatív lépésszáma *lineáris* (♣).

Ha a $\mathcal{P}$ párhuzamos és az $\mathcal{A}$ soros algoritmusra

$$\frac{W(n, \mathcal{A})}{W(n, p, \mathcal{P})} = o(p), \tag{1.20}$$

akkor $\mathcal{P}$ A-ra vonatkozó relatív lépésszáma *szublineáris* (♣).

Ha a $\mathcal{P}$ párhuzamos és az $\mathcal{A}$ soros algoritmusra

$$\frac{W(n, \mathcal{A})}{W(n, p, \mathcal{P})} = \omega(p), \tag{1.21}$$

akkor $\mathcal{P}$ A-ra vonatkozó relatív lépésszáma *szuperlineáris* (♣).

A párhuzamos algoritmusok esetében fontos jellemző az $m(n, p, \mathcal{P})$ *munka* (♣), amit a lépésszám és a processzorszám szorzatával definiálunk. Akkor is ez a szokásos definíció, ha a processzorok egy része csak a lépésszám egy részében dolgozik:

$$m(n, p, \mathcal{P}) = pW(n, p, \mathcal{P}). \tag{1.22}$$

Érdemes hangsúlyozni, hogy – különösen az aszinkron algoritmusok esetében – a ténylegesen elvégzett lépések száma lényegesen kevesebb lehet, mint amit a fenti (1.22) képlet szerint kapunk.

Egy $\mathcal{P}$ párhuzamos algoritmusnak az ugyanazon problémát megoldó soros algoritmusra vonatkozó $h(n, p, \mathcal{P}, \mathcal{A})$ *hatékonysága* (♣) a két algoritmus munkájának hányadosa:

$$h(n, p, \mathcal{P}, \mathcal{A}) = \frac{W(n, A)}{pW(n, p, \mathcal{P})} . \tag{1.23}$$

Abban a természetes esetben, amikor a párhuzamos algoritmus munkája legalább akkora, mint a soros algoritmusé, a hatékonyság nulla és egy közötti érték – és a viszonylag nagy érték a kedvező.

Központi fogalom a párhuzamos algoritmusok elemzésével kapcsolatban a munkahatékonyság. Ha a $\mathcal{P}$ párhuzamos és $\mathcal{A}$ soros algoritmusra

$$pW(n, p, \mathcal{P}) = O(W(n, \mathcal{A})), \tag{1.24}$$

akkor $\mathcal{P}$ $\mathcal{A}$-ra nézve *munkahatékony* (♣).

Ez a meghatározás egyenértékű a

$$\frac{pW(n, p, \mathcal{P})}{W(n, \mathcal{A})} = O(1) \tag{1.25}$$

egyenlőség előírásával. Eszerint egy párhuzamos algoritmus csak akkor munkahatékony, ha összes munkája nagyságrendileg nem nagyobb, mint a soros algoritmus munkája.

Egy párhuzamos algoritmus akkor és csak akkor munkahatékony, ha a relatív lépésszáma lineáris. Egy munkahatékony párhuzamos algoritmus hatékonysága $\Theta(1)$.

Ha egy algoritmus egy feladat megoldásához adott erőforrásból csak $O(N(n))$ mennyiséget használ fel, akkor az algoritmust az adott erőforrásra, számítási modellre (és processzorszámra) nézve *aszimptotikusan optimálisnak* (♣) nevezzük.

Ha egy $\mathcal{A}$ ($\mathcal{P}$) algoritmus egy feladat megoldásához adott erőforrásból a bemenet minden lehetséges $n \geq 1$ mérete esetében csak az okvetlenül szükséges $N(n, \mathcal{A})$ – illetve $(N(n, p, \mathcal{A}))$ – mennyiséget használja fel, azaz

$$W(n, \mathcal{A}) = N(n, \mathcal{A}), \tag{1.26}$$

illetve

$$W(n, p, \mathcal{P}) = N(n, p, \mathcal{P}), \tag{1.27}$$

akkor az algoritmust az adott erőforrásra (és az adott számítási modellre) nézve *abszolút optimálisnak* (♣) nevezzük és azt mondjuk, hogy a vizsgált feladat *pontos bonyolultsága* $N(n)$ $(N(n, p))$ (♣).

Két algoritmus összehasonlításakor a

$$W(n, \mathcal{A}) = \Theta(W(n, \mathcal{B})) \tag{1.28}$$

esetben azt mondjuk, hogy a $\mathcal{A}$ és $\mathcal{B}$ algoritmus lépésszámának növekedési sebessége aszimptotikusan *azonos nagyságrendű* (♣).

Amikor két algoritmus lépésszámát (például a legrosszabb esetben) összehasonlítjuk, akkor gyakran találunk olyan váltási helyeket, melyeknél kisebb méretű bemenetekre az egyik, míg nagyobb méretű bemenetekre a másik algoritmus lépésszáma kedvezőbb. A formális definíció a következő: ha a pozitív egész helyeken értelmezett $f(n)$ és $g(n)$ függvényekre, valamint a $v \geq 0$ pozitív egész számra teljesül, hogy

1. $f(v) = g(v)$;
2. $(f(v-1) - g(v-1))(f(v+1) - g(v+1)) < 0$,

akkor a v számot az $f(n)$ és $g(n)$ függvények *váltási helyének* (♣) nevezzük.

Például két mátrix szorzatának a definíció alapján történő és a Strassen-algoritmussal történő kiszámítását összehasonlítva (lásd például Cormen, Leiserson, Rivest és Stein többször idézett új könyvét) azt kapjuk, hogy kis mátrixok esetén a hagyományos módszer, míg nagy mátrixok esetén a Strassen-algoritmus az előnyösebb – egy váltási hely van, melynek értéke körülbelül 20.

1.3. Pszeudokód

Az algoritmusok formális leírására a következő, a Pascal és a C++ nyelvek elemeiből összeállított pszeudokódot használjuk.

1. Az algoritmus blokkszerkezetét elsősorban a tagolás tükrözi. Ezt a megoldást a programozási nyelvekben is használják – bár a használat szabályai nyelvenként változnak. A pszeudokódot lényegesen egyszerűsíti és áttekinthetőbbé teszi – értelmezése tapasztalataink szerint nem okoz gondot.

2. A változók neve betűvel kezdődik. A változók típusát külön nem deklaráljuk, mert az a környezetből látszik. Egyszerű adattípusok (mint egész, lebegőpontos, karakter, logikai stb.) fognak szerepelni.

3. A változók értékadó utasításokkal kapnak értéket:
$\langle \textit{változónév} \rangle := \langle \textit{kifejezés} \rangle$

4. Két logikai érték van, a **true** és a **false**. Ezek a logikai értékek az **and** ($\wedge$), **or** ($\vee$) és a **not** ($\neg$) logikai operátorokkal, valamint a $<$, $\leq$, $=$, $\geq$ és $>$ relációs operátorokkal állíthatók elő.

5. A többdimenziós tömbök elemei szögletes zárójelek közé tett indexekkel érhetők el, például $A[i, j]$. Az indexek nullától kezdődnek. A tömb egy részére az indextartomány megadásával hivatkozhatunk: például $A[3 : n]$.

6. A következő két ciklusszervező utasítást alkalmazzuk: **while** és **for**.
A **while** ciklus alakja a következő:

while $\langle$*feltétel*$\rangle$
$\langle$1. *utasítás*$\rangle$
$\langle$2. *utasítás*$\rangle$
.
.
.
$\langle$*u. utasítás*$\rangle$

Amíg a $\langle$*feltétel*$\rangle$ **true**, az u ($u \geq 1$) utasítás végrehajtódik. Amikor a $\langle$*feltétel*$\rangle$ **false** lesz, kilépünk a ciklusból.

A **for** ciklus alakja a következő:

for $\langle$*ciklusváltozó*$\rangle$:= $\langle$*kezdő érték*$\rangle$ **to** $\langle$*befejező érték*$\rangle$
$\langle$1. *utasítás*$\rangle$
$\langle$2. *utasítás*$\rangle$
.
.
.
$\langle$*u. utasítás*$\rangle$

Ha például a ciklusváltozó i, a kezdőérték k és a befejező érték b, akkor az u utasítás egymás után végrehajtódik az $i = k,\ k+1,\ \ldots, b$ értékekre.

7. A feltételes utasítás lehetséges alakjai:

if $\langle$*feltétel*$\rangle$
then
$\langle$1. *utasítás*$\rangle$
$\langle$2. *utasítás*$\rangle$
.
.
.
$\langle$*u. utasítás*$\rangle$

vagy

if ⟨*feltétel*⟩
    then
        ⟨1. *utasítás*⟩
        ⟨2. *utasítás*⟩
        .
        .
        .
        ⟨*u*. *utasítás*⟩
    else
        ⟨1. *utasítás*⟩
        ⟨2. *utasítás*⟩
        .
        .
        .
        ⟨*v*. *utasítás*⟩

8. A bevitel/kivitel a **read** és **write** utasításokkal történik. Pontos formájuk számítási modellenként változó.

9. Egyetlen eljárás van, amely fejből és törzsből áll. A fej

⟨E ´ ´ ⟩(⟨*paraméterlista*⟩) ⟨*eljárás típusa*⟩

Számítási modell: ⟨*számítási modell*⟩

Bemenet: ⟨*bemenő adatok leírása*⟩

Kimenet: ⟨*kimenő adatok leírása*⟩

Az eljárás típusa lehet *soros eljárás*, *soros rekurzív eljárás*, *párhuzamos eljárás* és *párhuzamos rekurzív eljárás*.

Az eljárások törzse sorszámozott utasításokból áll. Az utolsó utasítást kivéve az egyes utasítások végét a következő sorszám, a törzs végét a tagolás jelzi.

Ezek a számozott utasítások gyakran az algoritmus nagy (több lépésből álló) részét tükrözik. Ilyenkor az elemzésben ezeket a részeket szakasznak vagy fázisnak nevezzük. Más esetekben több számozott lépést együtt nevezünk az algoritmus szakaszának vagy fázisának.

10. A soros eljárások hívásának alakja:

call ⟨E ´ ´ ⟩⟨(*paraméterek listája*)⟩

A soros és a párhuzamos eljárások esetén egyaránt ezt a hívó utasítást használjuk.

11. A megjegyzések a ▷ jellel kezdődnek és az adott sor végéig tartanak.

12. A párhuzamosságot egy p-processzoros PRAM vagy lánc esetében a következőképpen írjuk le:

P_i **in parallel for** $1 \le i \le p$
⟨1. *utasítás*⟩
⟨2. *utasítás*⟩
.
.
.
⟨u. *utasítás*⟩

Egy k-dimenziós rács esetében a hasonló utasítás a

$P_{i_1, i_2, \ldots, i_k}$ **in parallel for** $1 \le i_1 \le m_1, \ \ldots, \ 1 \le i_k \le m_k$

sorral kezdődik.

1.4. Számítási modellek

Az algoritmusokat absztrakt vagy konkrét gépeken hajthatjuk végre. Ebben az alfejezetben először röviden bemutatjuk a párhuzamos számítógépek főbb típusait, azután az absztrakt gépek közül a párhuzamos közvetlen hozzáférésű gépekkel és a hálózatokkal foglalkozunk.

1.4.1. Számítógépek

Az elmúlt évtizedekben sok különböző párhuzamos számítógépet építettek, és számos próbálkozás történt rendszerezésükre.

Flynn 1972-ben az utasításáram és az adatáram alapján 4 csoportot különböztetett meg:

- SISD (Simple Instruction Strem – Simple Data Strem);
- SIMD (Simple Instruction Strem – Multiple Data Strem);
- MISD (Multiple Instruction Strem – Single Data Strem);
- MIMD (Multiple Instruction Strem – Multiple Data Strem).

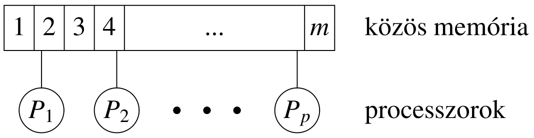
Eszerint a SISD a klasszikus soros, Neumann-elvű számítógép. A SIMD típusú számítógépben lépésenként egyféle művelet hajtódik végre, de az egyszerre több adaton. Az ILLIAC-IV számítógép példa erre a típusra.

A MISD típusú gépben egy adaton egyszerre többféle művelet hajtódik végre. A csővezeték-elvű számítógépek példák erre a típusra.

A legtöbb párhuzamos számítógép a MIMD típushoz tartozik: ebben az esetben több processzor dolgozik párhuzamosan, és rendszerint különböző adatokkal.

1.4.2. Párhuzamos gépek

A párhuzamos számítási modellek alapja a soros számításokhoz széles körben használt *RAM* (random access machine = *közvetlen hozzáférésű gép*) általánosítása, a *PRAM* (parallel random access machine = *párhuzamos közvetlen hozzáférésű gép*) (♣), p szinkronizáltan dolgozó processzorral (P_1, P_2, $\ldots, P_p$) és az $M[1]$, $M[2]$, $\ldots$, $M[m]$ rekeszekből álló közös memóriával, ahogy azt az 1.1. ábra mutatja (az ábrán a memóriarekeszeknek csak az indexét tüntettük fel). A modell szerint minden processzor rendelkezik saját memóriával: a P_i processzor esetében ez az $M[i,1]$, $M[i,2]$, $\ldots$, $M[i,m]$ rekeszekből áll. Nem jelent megszorítást, hogy a közös memória és a saját memóriák méretét ugyanúgy jelöltük (szokás ezeket a memóriákat végtelen méretűnek is tekinteni). Feltesszük, hogy a rekeszek tetszőleges egész szám tárolására alkalmasak.



1.1. ábra. Párhuzamos közvetlen hozzáférésű gép (PRAM)

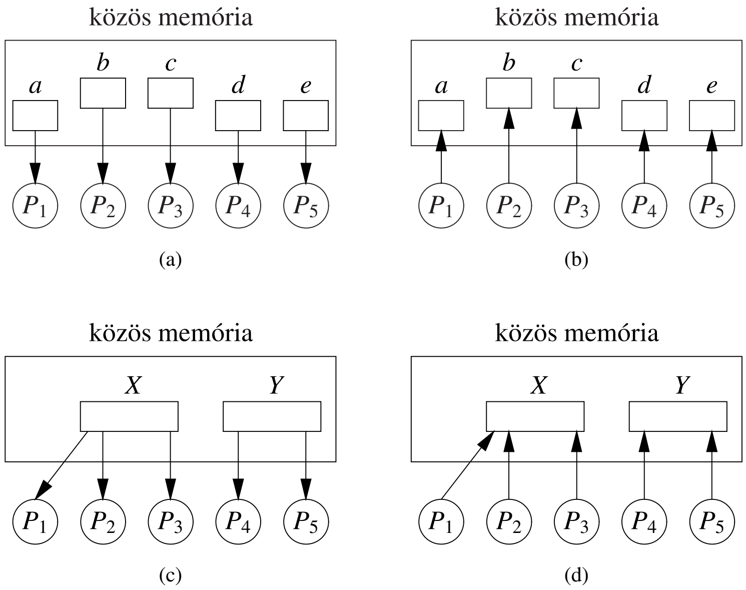
A párhuzamos közvetlen hozzáférésű gép helyett rendszerint a rövidebb *párhuzamos gép* kifejezést fogjuk használni.

Típusok írás/olvasás alapján:

- EREW (Exclusive Read – Exclusive Write: kizárólagos olvasás – kizárólagos írás)
- ERCW (Exclusive Read – Concurrent Write)
- CREW (Concurrent Read – Exclusive Write)
- CRCW (Concurrent Read – Concurrent Write)

Ugyanabba a memóriarekeszbe egyidejűleg csak írás vagy olvasás van megengedve.

Az 1.2. ábra (a) része arra példa, hogy minden rekeszből legfeljebb egy processzor olvas (ER), a (b) részében minden rekeszbe legfeljebb egy processzor ír



1.2. ábra. Számítási modellek típusa az írás és olvasás tulajdonságai alapján

(EW), a (c) részében több processzor olvas párhuzamosan (CR), végül a (d) részében több processzor ír egyidejűleg ugyanabba a rekeszbe (CW).

Ha több processzor írhat egyidejűleg (ERCW vagy CRCW), akkor több eset van: az írás

- *közös* (♣): a processzorok csak a közös, azonos értéket írhatják;
- *tetszőleges* (♣): nem tudjuk, melyik processzor beírása marad meg a rekeszben, vagy esetleg a beírásoktól független érték;
- *prioritásos* (♣): a legnagyobb prioritású processzor ír;
- *kombinált* (♣): az egyes processzorok által beírt értékek valamilyen függvénye kerül a memóriarekeszbe.

A következő példákban a párhuzamos olvasást, párhuzamos írást, illetve egy logikai érték párhuzamos kiszámítását mutatjuk be.

Először egy 4-processzoros gépben minden processzor a közös $M[1]$ rekeszből olvas.

1.1. példa. 4 processzor párhuzamosan olvas

P´ -O ($M[1]$) *párhuzamos eljárás*

Számítási modell: CREW PRAM

Bemenet: $A[1]$ (egy elemű tömb)

Kimenet: $M[i, 1]$ (processzorok saját memóriái első rekeszeinek közös tartalma)

P_i **in parallel for** $1 \leq i \leq 4$
  $M[i, 1]$:= **read** $A[1]$

♠

Ebben az esetben mind a 4 processzor saját memóriájának első rekeszébe bekerül az $A[1]$ tömbelem.

A következő példában egy 16-processzoros gépben minden processzor a közös $A[1]$ rekeszbe ír.

1.2. példa. 16 processzor párhuzamosan ír

P´ -´ ($M[1]$) *párhuzamos eljárás*

Számítási modell: közös ERCW PRAM

Bemenet: $M[1 : 16, 1]$ (16 elemű tömb)

Kimenet: $A[1]$ (tömb egy eleme)

P_i **in parallel for** $1 \leq i \leq 16$
  $A[1]$:= **write** $M[i, 1]$

♠

Ebben az esetben a 16 processzor párhuzamosan beírja az $A[1]$ tömbelembe saját memóriája első rekeszének tartalmát.

Legyen az $A[1:16]$ tömb $A[1], \ldots, A[n]$ elemeiben tárolt n bit logikai összege (diszjunkciója) $A[0] = A[1] \vee A[2] \vee \cdots \vee A[n]$.

Ekkor a következő ERCW algoritmus $O(1)$ időben dolgozik.

1.3. példa. Logikai összeg kiszámítása n processzoron

L -¨ $(p, A, A[0])$ *(párhuzamos eljárás)*

Számítási modell: ERCW PRAM

Bemenet: p (változók száma) és

$A[1..p]$ (a változókat tartalmazó tömb)

Kimenet: $A[0]$ (a bemenő változók logikai összege)

```
A[0] := 0
P_i in parallel for 1 ≤ i ≤ p
        if A[i] = 1
            then
                A[0] := A[i]
```

♠

1.4. tétel. (*Logikai összeadás művelet elvégzése.*) *A* L -¨ - *algoritmus az n-bites* $\vee$ *műveletet egy* ERCW PRAM-*en* $O(1)$ *lépés alatt elvégzi.*

Most a párhuzamos gépek néhány mennyiségi tulajdonságát mutatjuk be.

Feltesszük, hogy egy p-processzoros gép bármely lépése szimulálható egy soros processzor p lépésével (vannak olyan valódi gépek, amelyekre ez a feltétel nem teljesül). Ebből adódik a következő állítás.

1.5. tétel. (*Brent tétele.*) *Ha egy feladatot a* $\mathcal{P}$ *párhuzamos algoritmus* p *processzoron* t *lépésben old meg, eközben az* i*-edik* ($i = 1, 2, \ldots, t$) *lépésben* x_i *műveletet végez, akkor ez a feladat* $q < p$ *processzoron megoldható*

$$T + \left\lceil \frac{x}{p} \right\rceil \tag{1.29}$$

lépésben, ahol

$$x = \sum_{i=1}^{t} x_i. \tag{1.30}$$

Ebből a tételből adódik a következő egyszerű állítás.

1.6. következmény. (*Lassulás.*) *Ha egy* $\mathcal{P}$ *párhuzamos algoritmus egy* p*-processzoros gépen* t *lépést tesz, akkor minden* $q < p$ *processzort tartalmazó gépen végrehajtható* $O(\frac{pt}{q})$ *lépésben.*

Bizonyítás. A p-processzoros G gépen futó $\mathcal{P}$ párhuzamos algoritmus minden lépése szimulálható egy q-processzoros G gépen, legfeljebb $\lceil \frac{p}{q} \rceil$ lépést végezve. Ezért a H szimulációs lépésszáma legfeljebb $T\lceil \frac{p}{q} \rceil$, és így H munkája legfeljebb

$$qt \left\lceil \frac{p}{q} \right\rceil \leq pt + qT \tag{1.31}$$

$$= O(pt). \tag{1.32}$$

♠

A következő 3 állítás az elérhető relatív lépésszám mértékével kapcsolatos.

1.7. tétel. (*Amdahl törvénye a maximális relatív lépésszámról.*) *Ha egy* π *feladat megoldásának nem párhuzamosítható hányada* $s(\pi)$, *akkor egy* p*-processzoros* PRAM*-on elérhető* $g_{max}(\pi, s, p)$ *legnagyobb relatív lépésszám*

$$g_{max}(\pi, s, p) = \frac{1}{s + \frac{1-s}{p}}\ . \tag{1.33}$$

1.8. tétel. (*Gustafson törvénye a maximális relatív lépésszámról.*) *Ha egy π feladat megoldásának nem párhuzamosítható hányada s, akkor egy p-processzoros* PRAM-*on elérhető $g_{max}(\pi, s, p)$ legnagyobb relatív lépésszám*

$$\begin{aligned} g_{max}(\pi, s, p) &= \frac{s + p(1-s)}{s + (1-s)} && (1.34) \\ &= s + p(1-s). && (1.35) \end{aligned}$$

Amdahl és Gustafson tételének bizonyítását meghagyjuk gyakorlatnak.

Amdahl szerint s reciproka korlátot szab az elérhető relatív lépésszámnak, míg Gustafson szerint a relatív lépésszám a processzorszámmal arányosan növelhető.

1.9. tétel. (*Van-e p-nél nagyobb relatív lépésszám?*) *A p processzorszámnál nagyobb relatív lépésszám nem érhető el.*

Bizonyítás. Tegyük fel, hogy egy π problémára $W(n, \mathcal{A})$ a legjobb ismert soros végrehajtási idő. Ha van olyan $\mathcal{P}$ párhuzamos algoritmus, melynek relatív lépésszáma p-nél nagyobb, akkor $pW(n, \mathcal{P}, p) < W(n, \mathcal{A})$. Mivel egy p-processzoros gép egy lépésének szimulálása az 1-processzoros gépen legfeljebb p lépést igényel, ezért $\mathcal{P}$ munkája sorosan szimulálható legfeljebb $pW(n, \mathcal{P}, p)$ idő alatt, ami feltételünk szerint kisebb, mint $W(n, \mathcal{A})$. Ez ellentmond annak, hogy $\mathcal{A}$ a π megoldására ismert legjobb soros algoritmus. ♠

1.4.3. Hálózatok

A számítási modellek másik típusát a hálózatok adják. Ezekben a processzorok nem a közös memórián keresztül érintkeznek, hanem adatátviteli vonalakon keresztül, amelyek jól szemléltethetők gráfok segítségével. A processzor és a csúcs szavakat szinonimaként használjuk – általában a szövegkörnyezethez jobban illeszkedőt választva.

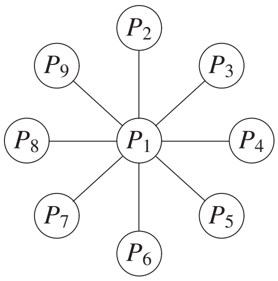
Ez a tény egyúttal ad egy jó szempontot a hálózatok osztályozására: síkba rajzolható és síkba nem rajzolható hálózatokat különböztetünk meg.

A gráfelmélet ismert eredménye, hogy minden véges gráf ábrázolható a 3-dimenziós euklideszi térben. Ezt beláthatjuk például úgy, hogy a $G = (V, E)$ véges gráf V_i ($i = 1,\ 2,\ \ldots, |V|$) csúcsát az x-y sík $(i, 0)$ pontjába rajzoljuk, majd a síkot az x-tengely körül rendre elforgatjuk $j\frac{360}{|V|}$ ($j = 1,\ 2,\ \ldots\ |V| - 1$) fokkal. Az így kapott S_j ($j = 1,\ 2,\ \ldots,\ |V| - 1$) síkokban rendre a V_j csúcsot a nála nagyobb indexű csúcsokkal összekötő éleket rajzoljuk meg.

A rövidség kedvéért a *síkba rajzolható* gráfokat *síkgráfoknak*, a síkba nem rajzolható gráfokat pedig *térgráfoknak* fogjuk nevezni. Ennek megfelelően beszélhetünk *síkhálózatról* (♣) és *térhálózatról* (♣).

A legismertebb hálózatok közül a síkhálózatokhoz tartozik például a csillag, fa, lánc, gyűrű, négyzet és a henger.

A p-processzoros *csillagban* (♣) kitüntetett szerepe van a P_1 processzornak: ez van minden további processzorral összekötve. Az 1.3. ábra egy 9-processzoros csillagot mutat.



1.3. ábra. 9-processzoros csillag

Egy d-szintes (teljes) bináris fában $p = 2^d - 1$ processzor van: $P_1, P_2, \ldots, P_p$. Az adatszerkezetekkel kapcsolatos terminológia szerint a P_1 processzort *gyökérnek*, a $P_{(p-1/2}$, $P_{(p-1)/2+1}$, $\ldots$, P_p processzorokat *levélnek*, a többi processzort *belső processzornak* nevezzük. Ha P_i nem levél, akkor össze van kötve a *gyerekeinek* nevezett P_{2i} és P_{2i+1} processzorokkal. Ha P_j nem a gyökér, akkor össze van kötve a *szülőjének* nevezett $P_{\lfloor j/2 \rfloor}$ processzorral. A negyedik fejezetben lévő 4.8. ábra egy 3-szintes bináris fa hálózatot ábrázol. Hasonlóképpen lehetne m-áris fákat és nem teljes fákat is definiálni.

A p-processzoros gyűrűs hálózatot a negyedik fejezetben ismertetjük. A 4.4. ábra egy 6-processzoros gyűrűt mutat.

A térhálózatok közül megemlítjük a k-dimenziós rácsot, kockát, tóruszt, piramist, pillangót, permutációs hálózatot, de Bruijn-hálózatot és a hiperkockát.

A k-dimenziós ($k \geq 1$) rács egy olyan $m_1 \times m_2 \times \cdots \times m_k$ ($m_1,\ m_2, \ldots,\ m_k \geq 2$) méretű háló, amelynek minden egyes metszéspontjában van egy processzor. Az élek a kommunikációs vonalak, melyek kétirányúak. A rács minden processzorát megcímkézzük egy $(i_1, i_2, \ldots i_k)$ k-assal – erre a processzorra a $P_{i_1,\ldots,i_k}$ jelöléssel hivatkozunk.

Minden processzor egy RAM, amely rendelkezik saját (helyi) memóriával. A $P_{i_1,\ldots,i_k}$ processzor saját memóriája az $M[i_1, \ldots, i_k, 1]$, $M[i_1, \ldots, i_k, 2]$, $\ldots$, $M[i_1, \ldots, i_k, m]$ rekeszekből áll.

Minden processzor végre tud hajtani egyetlen lépésben olyan alapvető műveleteket, mint az összeadás, kivonás, szorzás, összehasonlítás, saját memória elérése és így tovább. A processzorok működése szinkron módon történik, azaz minden processzor egy globális óra ütemére egyszerre hajtja végre az aktuális feladatát.

A legegyszerűbb rács a $k = 1$ értékhez tartozó lánc alakú rács (röviden *lánc*) (♣).

Egy 6-processzoros lánc látható a harmadik fejezetben lévő 3.1. ábrán. Ha egy lánc P_1 és P_p processzorát összekötjük, akkor gyűrűt kapunk.

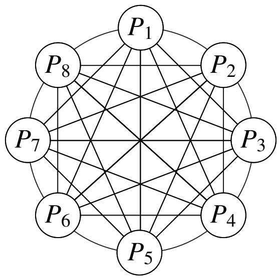
Ha $k = 2$, akkor *téglalapot* (téglalap alakú rácsot) kapunk. Ha most $m_1 = m_2 = \sqrt{p}$, akkor $a \times a$ méretű *négyzetet* kapunk. Egy 4×4 méretű négyzet látható a harmadik fejezetben lévő 3.2. ábrán.

A lánc és a négyzet a síkhálózatokhoz tartoznak.

Ha egy rácsot további élekkel kiegészítünk, akkor *összetett* rácsot kapunk.

Ha egy láncban a P_1 és P_p processzorokat összekötjük, akkor gyűrűt kapunk, amely már csak két dimenzióban ábrázolható A negyedik fejezetben lévő 4.4. ábra egy 6-processzoros láncot ábrázol.

Ha egy téglalapban a sorok első ($j = 1$) és utolsó ($j = m_2$) elemeit összekötjük, akkor az ugyancsak 2-dimenziós *hengert* kapjuk. Az 1.4. ábra egy 4×4 méretű hengert ábrázol.



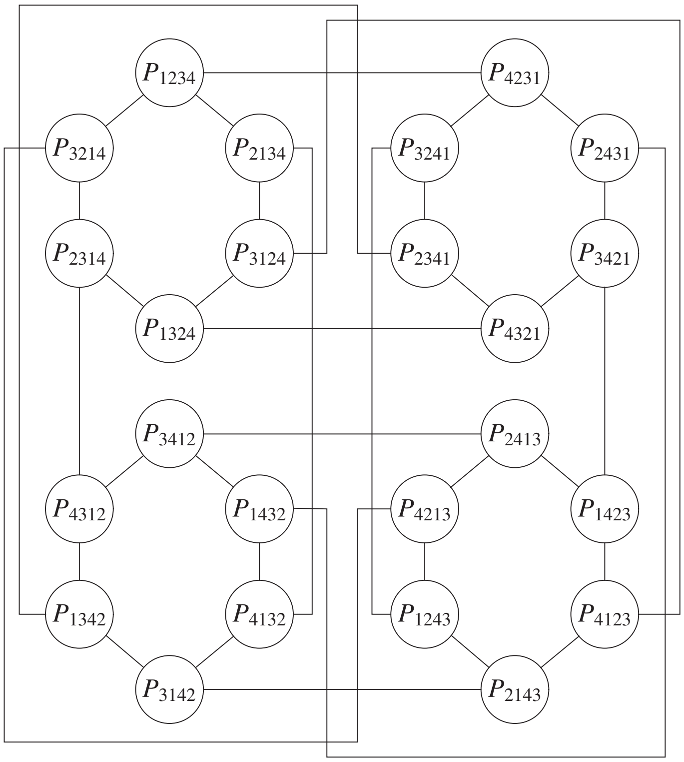
1.4. ábra. 4×4 méretű henger

Az ILLIAC-IV megépült változata olyan 2-dimenziós rácsból származtatható, amelyben $m_1 = m_2 = 8$ és a $P_{i,j}$ processzor szomszédai rendre a $P_{i,j-1}$, $P_{i,j+1}$ $P_{i+1,j}$, $P_{i-1,j}$, ahol az indexek (mod 8) értendők. Az ILLIAC-IV architektúrájának ábrázolásához már három dimenzióra van szükség.

Ha $k = 3$ és $m_1 = m_2 = m_3$, akkor *téglát* kapunk. Ha ennek a hálózatnak a 3 mérete azonos, akkor *kockának* nevezzük. A harmadik fejezetben lévő 3.3. ábra egy $2 \times 2 \times 2$ méretű kockát (♣) ábrázol.

A *tórusz* (♣) például úgy származtatható egy 2-dimenziós rácsból, hogy a belső élek mellett az egyes sorok első és utolsó processzorait, valamint az egyes oszlopok első és utolsó oszlopait is összekötjük. A negyedik fejezetben lévő 4.6. ábra egy 5×5 méretű tóruszt mutat.

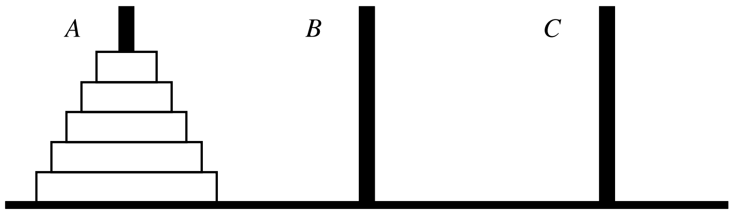
Egy *d-szintes piramis* (♣) *i*-edik ($1 \leq i \leq d$) szintjén 4^{d-i-1} processzor van, amelyek az 1.5. ábra szerint vannak összekötve. Eszerint a piramis *i*-edik szintje egy $2^{d-i} \times 2^{d-i}$ méretű rács. A piramis az egyes szinteken lévő processzorok számát tekintve hasonló a ternáris fához, azonban a fa azonos szinten lévő processzorai között nincs kapcsolat.



1.5. ábra. 3-szintes piramis

A *d-dimenziós pillangó* hálózat $(d + 1)2^d$ processzorból áll, amelyek $d + 1$ – egyenként 2^d processzort tartalmazó – sorban vannak elrendezve, ahogy azt a negyedik fejezetben lévő 4.2. ábra mutatja. A $(d+1)\times 2^d$ méretű pillangó kifejezést is használni fogjuk.

Természetes architektúra a *teljes hálózat* (♣), amelyben minden processzor pár össze van kötve. Egy teljes hálózatot ábrázol az 1.6. ábra.

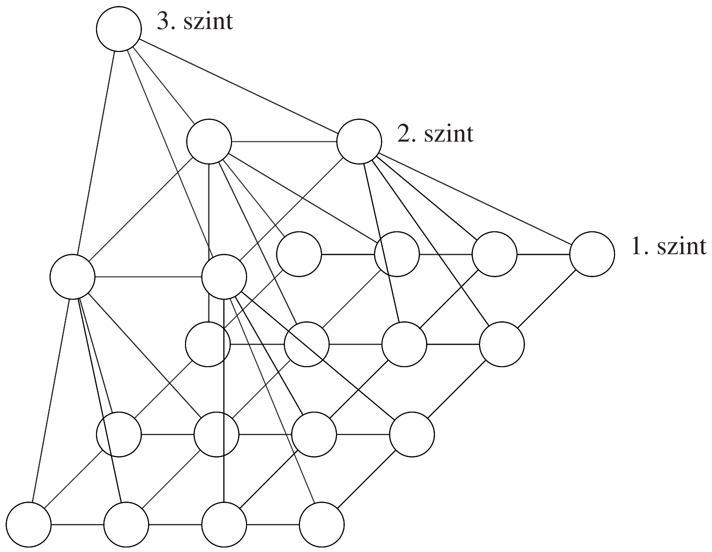


1.6. ábra. 6-processzoros teljes hálózat

A (d, k)-paraméterű *de Bruijn-hálózat* (♣) d^k processzort tartalmaz, amelyek a d betűs $\{0, 1, \ldots, d-1\}$ ábécé feletti k hosszúságú szavakkal címezhetők. Az $a_1a_2\ldots a_k$ nevű processzorból az $a_2a_3\ldots a_kq$ című processzorokba vezet irányított él – ahol q az ábécé tetszőleges eleme. Eszerint minden processzorból d él indul, és minden processzorban d él végződik. Az 1.7. ábra egy (2,3)-paraméterű de Bruijn-hálózatot mutat. Az ábra alapján ez 2-dimenziós.

A d-dimenziós *permutációs hálózatban* a processzorok a d-betűs $\{0, 1, \ldots, d-1\}$ ábécé elemeinek különböző permutációival címezhetők. Ezért a hálózatban $p = d!$ processzor van. A P_i processzor pontosan azokkal a processzorokkal van összekötve, amelyek címkéje előállítható úgy, hogy P_i címkéjében az első betűt a j-edik $(2 \leq j \leq d)$ betűvel cseréljük ki. A 1.8. ábra egy 4-paraméterű permutációs hálózatot ábrázol. Ebben minden processzor fokszáma $d-1 = 3$ és a processzorok száma $4! = 24$.

A d-dimenziós hiperkockában 2^d processzor van, amelyek d hosszúságú bináris sorozatokkal címezhetők. Minden P_i processzor pontosan azokkal a processzorokkal van összekötve, amelyek címe pontosan egy bitben tér el P_i címétől.



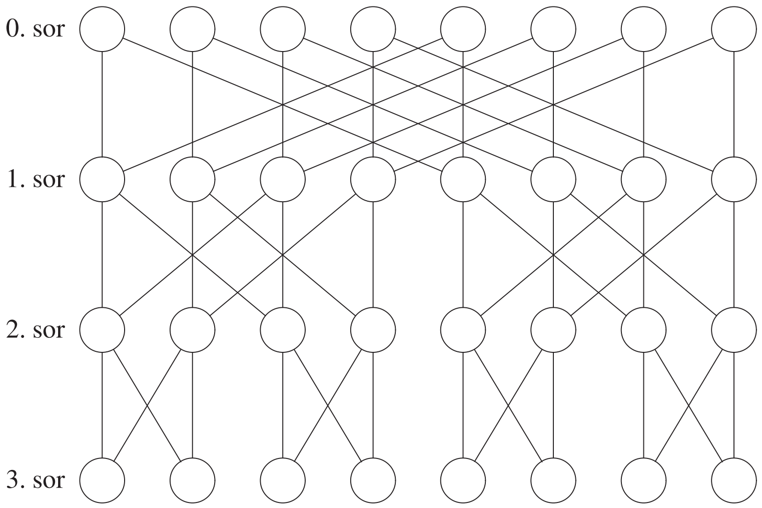
1.7. ábra. 2×3 paraméterű de Bruijn-hálózat

A 3-dimenziós hiperkockában a 0,1,0 csúcs szomszédjai az 1,1,0, 0,0,0 és a 0,1,1 című csúcsok. Mivel a $2\times2\times2$ méretű kocka egyúttal 3-dimenziós hiperkocka is, a negyedik fejezetben lévő 3.3. ábra egyúttal egy 3-dimenziós hiperkockát is bemutat. Az ugyancsak a negyedik fejezetben lévő 4.1. ábra pedig egy 4-dimenziós hiperkockát mutat.

A d-dimenziós rácsok, permutációs hálózatok és hiperkockák természetes módon elképzelhetők és ábrázolhatók d dimenzióban. A korábban említett tétel szerint ugyanakkor tetszőlegesen nagy d esetében is ábrázolhatók a 3-dimenziós euklideszi térben úgy, hogy az élek ne metsszék egymást.

A hálózatok jellemzésére sokféle adatot használnak. Ha a H hálózatot $H = (V, E)$ formában, a csúcsok és élek halmazával adjuk meg, akkor természetes adat a *processzorok* $|V|$ (♣) és az *adatátviteli vonalak* $|E|$ (♣) száma.

Az adatátviteli vonalak lehetnek egyirányúak és kétirányúak. Az eddigi pél-



1.8. ábra. 24-processzoros permutációs hálózat

dák közül csak a de Bruijn-hálózat tartalmazott egyirányú éleket (ezért a hálózatot irányított gráffal ábrázoljuk), míg a többi esetben kétirányú éleket tételeztünk fel.

További jellemző a csúcsok *maximális* (♣), *minimális* (♣) és *átlagos fokszáma*, (♣) a csúcsok *maximális* (♣), *minimális* (♣) és *átlagos távolsága* (♣).

A csúcsok maximális távolságát a hálózat *átmérőjének* (♣) nevezik.

Ha egy $H = (V, E)$ összefüggő hálózat csúcsait X és Y halmazra osztjuk, akkor a hálózat adott felbontáshoz tartozó *vágási száma* (♣) a legkisebb olyan élhalmaz elemszáma, amellyel a hálózat elveszti összefüggőségét.

Hálózatok *felezési száma* (♣) azon élek minimális száma, amelyek eltávolításával a hálózat két azonos részre bontható. Ha egy hálózat processzorainak száma páros ($p = 2k$), akkor a hálózat biztosan felbontható két azonos részre (például két, egyenként k izolált csúcsot tartalmazó hálózatra). Ha $p = 2k + 1$, akkor a hálózat nem bontható két azonos részre. Ha $\sqrt{p}$ páros, akkor egy $\sqrt{p} \times \sqrt{p}$ méretű rács vágási száma $\sqrt{p}$.

Az 1.2. táblázat az ismertetett hálózatok néhány alapvető adatát tartalmazza. A táblázat *Paraméter* oszlopában p a processzorszámot jelöli. d a fa és a piramis esetében a szintek számát, permutációs hálózat, de Bruijn hálózat és hiperkocka esetében a dimenziót jelöli. Négyzet és kocka esetében az oldalhosszúság a paraméter.

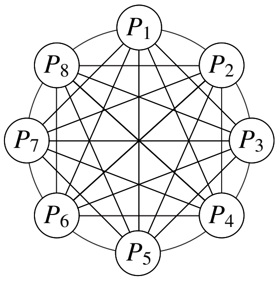
1.2. táblázat. Hálózatok mennyiségi jellemzői

Hálózat	Paraméter	Csúcsszám	Élszám	Átmérő
Lánc	p	p	$p-1$	$p-1$
Gyűrű	p	p	p	$\lceil \frac{p}{2} \rceil$
Csillag	p	p	$p-1$	1
Fa	d	2^d-1	$2d-2$	$2d-2$
Négyzet	$a = \sqrt{p}$	a^2	$2d(d-1)$	$2d-2$
Kocka	$a =^3 \sqrt{p}$	a^3	$2(a-1)a^2$	$3a-3$
Piramis	d	$\frac{4^{d+1}-1}{3}$	$2d$	2^d-2
Permutációs	d	d^2	$2d$	$\frac{d(d-1)!}{2}$
De Bruijn	d	2^d	2^{d+1}	d
Hiperkocka	d	2^d	$d2^{d-1}$	d

1.5. Rekurzió

Az algoritmusok megadásának gyakran alkalmazott módja az, hogy a végrehajtás során – változó paraméterekkel – újra és újra alkalmazzuk magát az algoritmust. Egy algoritmusnak önmaga segítségével való megadását *rekurziónak*, az így megadott algoritmust pedig *rekurzív algoritmusnak* nevezzük.

Példaként oldjunk meg egy népszerű, 1883-ból származó, Hanoi híres tornyairól szóló feladatot, melyben korongok és 3 rúd szerepel. Brahma tornya 64 – közepén lyukas – arany korongból áll, melyet Brahma egy gyémánt rúdra tett. A korongok mérete alulról felfelé csökken. Az 1.9. ábra a kiinduló helyzetet mutatja 5 koronggal, melyek az A rúdon vannak. A B és C rudakon kezdetben nincs korong.



1.9. ábra. Hanoi tornyai 3 rúddal és 5 koronggal

A Világ teremtésével egyidejűleg Brahma megbízott szerzeteseket azzal, hogy az egyik rúdon lévő korongokat helyezzék át egy másik rúdra – egy harmadik rúd felhasználásával. A szerzetesek munkájának egy lépése egy korong áthelyezése egyik rúdról egy másik rúdra – azzal a megszorítással, hogy korongot csak üres rúdra vagy nála nagyobb korongra szabad helyezni. A történet szerint amikor a szerzetesek végeznek a munkával, a torony összeomlik és vége a világnak.

Mennyi időnk van hátra, ha a szerzetesek a lehető legkevesebb lépésben elvégzik a munkát – és egy lépés egy másodpercig tart.

Jelöljük $f_K(n)$-nel az n korong áthelyezéséhez szükséges lépések számát egy K korong-áthelyező algoritmus esetében. Minden K algoritmusra igaz, hogy

$$f_K(1) \geq 1. \tag{1.36}$$

n korongot csak úgy tudunk áthelyezni, hogy előbb a felső $n-1$ korongot áthelyezzük egy másik rúdra, ezután a legalsó korongot áthelyezzük, és végül az $n-1$ kisebb korongot ráhelyezzük a legnagyobb korongra. Eszerint minden K algoritmusra

$$f_K(n) \geq 2f_K(n-1) + 1. \tag{1.37}$$

Az ehhez hasonló egyenleteket – amelyekben egy függvény adott helyen felvett értékét más helyen vagy helyeken felvett értékei segítségével adjuk meg – *rekurzív egyenletnek* (♣) nevezzük.

Tegyük fel, hogy a szerzetesek a következő rekurzív program szerint dolgoznak, melyben az Á $(k, végez)$ eljárás feladata az, hogy a k nevű rúdon lévő legfelső korongot áthelyezze a *végez* nevű rúdra.

H - (*n,k,segít,végez,lépés*) *soros rekurzív eljárás*

Számítási modell: speciális

Bemenet: n (a korongok száma)

Kimenet: *lépés* (lépésszám n korong áthelyezése során)

```
lépés := 0
if n = 1 then
          call Á (k, végez)
          lépés := lépés + 1
     else
          call H - (n − 1, k, végez, segít, lépés)
          call Á (k, végez)
          lépés := lépés + 1
          call H - (n − 1, segít, k, végez, lépés)
```

Eszerint a szerzetesek először – $f_{Sz}(1)$ lépés alatt – átrakják a legkisebb korongot a B rúdra, majd – $f_{Sz}(2) = 2f_{Sz}(1) + 1 = 3$ lépés alatt – átrakják a két legkisebb korongot a C rúdra, majd – $f_{Sz}(3) = 2f_{Sz}(2) + 1$ lépés alatt – átrakják a három legkisebb korongot a B rúdra és így tovább. Ebből következik, hogy

$$f_{Sz}(n) = 2f_{Sz}(n-1). \tag{1.38}$$

Mivel a szerzetesek minden részfeladatot a lehető legkevesebb lépésben megoldanak, algoritmusuk optimális. Módszerük lépésszámát egyszerűen $f(n)$-nel jelölve a következő *kezdeti feltételt* (♣) és rekurzív egyenletet kapjuk:

$$f(1) = 1 \tag{1.39}$$

és

$$\text{ha } n \geq 2, \text{ akkor } f(n) = 2f(n-1) + 1. \tag{1.40}$$

Függvényeknek kezdeti feltétel (vagy feltételek) és rekurzív egyenlet segítségével történő megadását *rekurziónak* (♣) nevezzük. Eszerint a rekurzió szót kétféle – egymással összefüggő – jelentéssel használjuk.

Teljes indukcióval és a rekurziótétellel (lásd például Halmos könyvét) bebizonyíthatjuk, hogy ennek a rekurzív egyenletnek a megoldása az

$$f(n) = 2^n - 1 \tag{1.41}$$

függvény.

Eszerint a világ élettartama

$$\begin{aligned}
2^{64} - 1 \text{ másodperc} &\approx 1.8447 \times 10^{19} \text{ másodperc} && (1.42)\\
&\approx 3.0745 \times 10^{17} \text{ perc} && (1.43)\\
&\approx 5.1242 \times 10^{15} \text{ óra} && (1.44)\\
&\approx 2.1351 \times 10^{14} \text{ nap} && (1.45)\\
&\approx 5.8495 \times 10^{11} \text{ év.} && (1.46)
\end{aligned}$$

Tehát a korongok átrakása a szerzeteseknek körülbelül ötszáz milliárd évig tart. Eszerint még akkor is sok időnk lenne hátra a világ végéig, ha a szerzetesek

4,6 milliárd évvel ezelőtt (amikor a geológusok szerint a Föld és a világegyetem az ősrobbanással létrejött) kezdték volna el a korongok átrakását.

Ez a példa több szempontból nagyon érdekes.

Mutatja, hogy – legalábbis bizonyos esetekben – elemi eszközökkel meg tudjuk határozni a rekurzív algoritmusok lépésszámát.

Tegyük fel, hogy sok szerzetes dolgozik a korongok átrakásán, és s szerzetes már $\frac{1}{s}$ másodperc alatt átrak egy korongot, vagy pedig a lépéseket egyenletesen fel tudják egymás között osztani. Ekkor az algoritmus futási ideje az

$$f(n, s) = \frac{2^n - 1}{s} \tag{1.47}$$

értékre csökken. Ezzel elvben tetszőleges kicsire csökkenthető az átrakási idő. De hogyan tud a sok szerzetes hozzáférni a korongokhoz?

A kérdés rávilágít arra, hogy egy feladat részfeladatokra bontásának lehetőségeit gyakran korlátozzák a feladat sajátosságai.

Ha például megengedjük, hogy a szerzetesek felemeljék a felső $n-1$ korongot és a legalsót körcikkekre vágva áthelyezzék, akkor megfelelő számú szerzetes esetében lineáris (vagy akár konstans) futási idejű algoritmust kaphatunk. Ezzel a megközelítéssel a könyvben nem foglalkozunk: a párhuzamos processzorok csak olyan műveleteket végezhetnek, amelyet a soros is végre tudott hajtani.

Másik kérdés, mit érünk azzal, ha a gyakorlatilag végtelen lépésszámot például század vagy ezred részére csökkentjük? Ez a kérdés pedig arra világít rá, hogy a sokprocesszoros rendszerek és párhuzamos algoritmusok lehetőségei is korlátozottak.

A feladat megoldása azt is mutatja, hogy ha csak a lépésszámra van szükségünk, az (1.41) képletet levezetve és alkalmazva nagyon gyorsan meg tudjuk mondani.

1.6. Véletlenített algoritmusok (⋆)

A véletlenített algoritmusok bizonyos helyzetekben több döntés közül választanak – ezek valószínűsége pontosan adott.

Két típusuk van. A *Las Vegas algoritmusok* mindig helyes eredményt adnak – futási idejük azonban nem állandó, hanem egy valószínűségi változóval jellemezhető.

A *Monte Carlo algoritmusok* adhatnak hibás eredményt is.

A Las Vegas algoritmusok futási ideje tág határok között változhat, míg a Monte Carlo algoritmusoké viszonylag állandó.

A véletlenített algoritmusokat tekinthetjük algoritmushalmaznak. Adott bemenet esetében bizonyos algoritmusok futhatnak sokáig vagy adhatnak hibás választ. A tervezés célja, hogy az ilyen *rossz* (♣) algoritmusok hányada kicsi legyen. Ha meg tudjuk mutatni, hogy *bármilyen* bemenetre az algoritmusoknak legalább $1 - \epsilon$ hányada (ahol a pozitív ϵ elég kicsi) gyors (illetve helyes eredményt ad), akkor a halmazból véletlenül választott algoritmusra igaz, hogy legalább $1 - \epsilon$ valószínűséggel gyors (helyes eredményt ad). Ekkor ϵ-t *hibavalószínűségnek* (♣) nevezzük.

Az alábbi definíciókban a d, v és g függvények a pozitív egészek halmazán vannak értelmezve, a $d(n)$, $v(n)$ és $g(n)$ függvényértékek pedig nemnegatív valós számok.

Egy D determinisztikus algoritmusnak adott erőforrásból $d(n)$ egységre van szüksége, míg a V véletlenített algoritmusnak $v(n)$ egységre – ahol $v(n)$ valószínűségi változó.

Egy n-től függő ϵ esemény *nagy valószínűséggel* (♣) bekövetkezik, ha tetszőleges α értékre a bekövetkezés valószínűsége legalább $1 - n^{-\alpha}$. Az α számot *valószínűségi paraméternek* (♣) nevezzük.

$\overline{O}(g(n))$ (ejtsd: *nagy valószínűséggel nagy ordó*) (♣) azon $v(n)$ függvények halmazát jelenti, amelyekhez létezik olyan c pozitív állandó, hogy tetszőleges α esetében nagy valószínűséggel teljesül

$$P[v(n) \leq c\alpha g(n)] \geq 1 - \frac{1}{n^\alpha}. \tag{1.48}$$

1.6.1. Egyenlőtlenségek

Ebben a részben olyan egyenlőtlenségeket adunk meg, amelyek azt jellemzik, hogy a valószínűségi változók ritkán térnek el lényegesen a várható értéküktől.

Ezek az egyenlőtlenségek majd hasznosak lesznek a véletlenített algoritmusok várható viselkedésének elemzése során.

1.10. lemma. (*Markov-egyenlőtlenség.*) *Ha X nemnegatív valószínűségi változó, melynek várható értéke μ, akkor*

$$P[X \geq x] \leq \frac{\mu}{x}\,. \tag{1.49}$$

A *Bernoulli kísérletnek* (♣) 2 lehetséges eredménye van: p valószínűséggel sikeres, $1-p$ valószínűséggel sikertelen a kísérlet. Ha X jelöli azt, hogy n kísérletből hány sikeres, akkor X eloszlása binomiális:

$$P[X = i] = \binom{n}{i} p^i (1-p)^{n-i}. \tag{1.50}$$

1.11. lemma. (*Csernov-egyenlőtlenségek.*) *Ha X (n, p) paraméterű binomiális eloszlású valószínűségi változó és $m > np$ egész szám, akkor*

$$P[X \geq m] \leq \left(\frac{np}{m}\right)^m e^{m-np}\,. \tag{1.51}$$

Továbbá minden $0 < \epsilon < 1$ számra

$$P[X \leq \lfloor (1-\epsilon)pn \rfloor] \leq e^{-\epsilon^2 np/2} \tag{1.52}$$

és

$$P[X \geq \lceil (1+\epsilon)np \rceil] \leq e^{-\epsilon^2 np/3}\,. \tag{1.53}$$

1.6.2. Példák

Ebben a pontban két példát mutatunk véletlenített algoritmusra.

1.12. példa. Ismétlődő tömbelem meghatározása

Tegyük fel, hogy n páros, az $A[1:n]$ tömb elemei között az $1, 2, \ldots, \frac{n}{2}+1$ számok egyike $\frac{n}{2}$-ször fordul elő, a többi szám pedig egyszer.

Határozzuk meg a többször előforduló elemet.

Bármely determinisztikus algoritmusnak legrosszabb esetben legalább $\frac{n}{2}+2$ lépésre van szüksége. Ugyanis bármely determinisztikus algoritmushoz megadható úgy a bemenet, hogy az első $\frac{n}{2}+1$ vizsgált elem különböző legyen.

Most megadunk egy Las Vegas algoritmust, amely legrosszabb esetben $\overline{O}(\lg n)$ ideig fut. Az algoritmus bemenete egy n szám és egy n-elemű A tömb, kimenete az egyik ismételt elem indexe.

I ´ - $(A, n, ismételt)$ *soros eljárás*

Számítási modell: RAM

Bemenet: n pozitív egész (az elemek száma) és

$A[1:n]$ (a vizsgálandó elemek)

Kimenet: *ismételt* (az egyik ismételt elem indexe)

L := **true**
while L
  i := V´ (n)
  j := V´ (n)
  $\triangleright$ $i, j \in \{1, 2, \ldots, n\}$, véletlen egész számok
  if $i \neq j \wedge (a[i] = a[j])$ **then**
    ismételt := i
    L := **false**

Ebben az algoritmusban felhasználjuk a V´ (n) eljárást, amely minden hívásakor a $[1 : n]$ intervallumból vett egyenletes eloszlású, véletlen egész számot ad kimenetként.

Most megmutatjuk, hogy az algoritmus futási ideje $\overline{O}(\lg n)$. A **while** ciklus bármely végrehajtása sikeres, ha i az $\frac{n}{2}$ azonos elem valamelyikének indexe és j egy tőle különböző helyen lévő azonos elem indexe. Ennek az eseménynek a P valószínűsége

$$P = \frac{\frac{n}{2}(\frac{n}{2} - 1)}{n^2}, \tag{1.54}$$

ami $n > 10$ esetében kisebb, mint $\frac{1}{5}$. Tehát annak valószínűsége, hogy az algoritmus egy adott lépésben nem fejeződik be, $\frac{1}{5}$-nél kisebb.

Ezért annak valószínűsége, hogy az algoritmus 10 lépés alatt nem fejeződik be, $(\frac{4}{5})^{10}$-nél kisebb, ami kisebb 0.1074-nél. Ezért az algoritmus 10 lépés alatt legalább 0.8926 valószínűséggel befejeződik. Annak valószínűsége, hogy 100 lépés alatt sem fejeződik be, kisebb $(\frac{4}{5})^{100}$-nál, ami pedig kisebb $2.04 * 10^{-10}$-nél.

Általában annak az ϵ eseménynek a valószínűsége, hogy az algoritmus az első $c\alpha \lg n$ (ahol a c konstans rögzített érték) lépésben nem fejeződik be,

$$P[\epsilon] < \left(\frac{4}{5}\right)^{c\alpha \lg n}, \tag{1.55}$$

és

$$\text{ha } c \geq \frac{1}{\lg \frac{5}{4}}, \text{ akkor } \left(\frac{4}{5}\right)^{c\alpha \lg n} < n^{-\alpha}. \tag{1.56}$$

Tehát az algoritmus legalább $1 - n^{-\alpha}$ valószínűséggel befejeződik legfeljebb

$$\frac{\alpha \lg n}{\lg \frac{5}{4}} \tag{1.57}$$

lépésben. Mivel minden iteráció $O(1)$ ideig tart, ezért az algoritmus futási ideje valóban $\overline{O}(\lg n)$.

Mivel algoritmusunk mindig helyes eredményt ad, ezért Las Vegas típusú. Megmutattuk, hogy nagy valószínűséggel gyorsan befejeződik.

Ha például kétmillió elemünk van, akkor bármely $\mathcal{D}$ determinisztikus algoritmushoz megadhatunk olyan bemenetet, amelyre $\mathcal{D}$ legalább egymillió lépést végez, Az I ´ - algoritmusnak viszont bármely bemenetre – nagy valószínűséggel – legfeljebb száz lépésre van szüksége.

Ugyanakkor az I ´ - algoritmus – igaz, csak kis valószínűséggel – egymilliónál lényegesen több lépést is végezhet.

Az adott problémát másképp is megoldhatjuk: először rendezünk, azután lineárisan keresünk – ekkor $\Omega(n)$ időre van szükség. Vagy hármas csoportokra osztjuk a tömböt – ekkor $\Theta(n)$ a lépésszám. ♠

1.13. példa. Pénzérme ismételt feldobása

Dobjunk fel egy szabályos pénzérmét ezerszer. Mekkora annak valószínűsége, hogy legalább hatszázszor fejet dobunk?

A Markov-egyenlőtlenség szerint legfeljebb 5/6.

Az (1.53) képletet, azaz a harmadik Csernov-egyenlőtlenséget is alkalmazhatjuk az $n = 1000$, $p = 1/2$, $\epsilon = 0.2$ értékekkel. Eszerint

$$P[X \geq 600] \leq e^{-(0,2)^2(500/3)} \tag{1.58}$$

$$= e^{-20/3} \tag{1.59}$$

$$\leq 0.001273. \tag{1.60}$$

Meghagyjuk gyakorlatnak az ennél pontosabb becslést. ♠

1.7. Alsó korlátok

Az algoritmusok elemzése során számos korábbi tankönyv szerzői (a múlt század hetvenes és a nyolcvanas éveiben) megelégedtek azzal, hogy többé-kevésbé pontos felső korlátokat adtak az adott algoritmus által igényelt erőforrások legnagyobb és átlagos mennyiségére.

Ezeknek a becsléseknek a pontossága azonban gyakran homályban maradt. Pedig e nélkül megválaszolatlan marad az a fontos kérdés, hogy érdemes-e jobb algoritmust keresni.

Különösen fontos az erőforrásigény pontos ismerete a párhuzamos algoritmusok vizsgálata során, hiszen e nélkül nem tudjuk eldönteni, hogy adott párhuzamos algoritmus munkahatékony-e.

A munkahatékonyság bizonyításához a párhuzamos algoritmus igényét felülről, a feladat jellegéből fakadó igényt pedig alulról kell becsülnünk.

A munkahatékonyság cáfolásához pedig elég egy jó soros algoritmus igényének felülről, a párhuzamos algoritmus igényének pedig alulról való becslése.

Ebben az alfejezetben olyan módszereket mutatunk be, melyek segítségével alsó korlátokat bizonyíthatunk. Ez az alfejezet bevezető jellegű. A későbbiekben a számítási modellekhez és problémákhoz kapcsolódva további alsó korlátokat adunk meg.

1.7.1. Egyszerű számolás

Tegyük fel, hogy adott egy n méretű $L[1:n]$ lista és feladatunk a lista legnagyobb elemének megkeresése úgy, hogy elempárok összehasonlítása a megengedett művelet.

Ha $A(n)$-nel jelöljük az A kereső algoritmus által elvégzett összehasonlítások számát, akkor tetszőleges A összehasonlítás alapú algoritmusra teljesül, hogy

$$A(n) \geq N(n) \tag{1.61}$$

$$\geq \left\lceil \frac{n}{2} \right\rceil . \tag{1.62}$$

Megmutatjuk, hogy ennél több is igaz.

Az egyszerűség kedvéért tegyük fel, hogy a lista különböző elemeket tartalmaz. Amikor egy A összehasonlítás alapú algoritmus összehasonlítja az X és Y elemet, akkor az $X > Y$ esetben azt mondjuk, hogy X megnyerte az összehasonlítást – ellenkező esetben pedig azt mondjuk, hogy X elvesztette az összehasonlítást. Tehát minden összehasonlítás egy vereséget eredményez. Mivel mindazon elemeknek, amelyek nem a legnagyobbak, legalább egy összehasonlítást el kell veszíteniük, ezért n elem legnagyobbikának meghatározásához

$$B_A(n) \geq n - 1 \tag{1.63}$$

összehasonlításra van szükség.

Ennek szigorú bizonyításához tegyük fel, A úgy fejezi be a keresést, hogy az $L[1 : n]$ lista két különböző eleme, például $L[i]$ és $L[j]$, egyetlen összehasonlítást sem veszítettek el. Feltehetjük, hogy A az $L[i]$ elemet adta meg legnagyobbként. Most tekintsük azt az $L'[1 : n]$ bemenő listát, amely csak annyiban különbözik az előzőtől, hogy abban $L'[j]$ nagyobb, mint $L[j]$. Mivel A összehasonlítás alapú, ezért ugyanolyan összehasonlításokat végez L és L' esetében. Ez következik abból, hogy különbség legfeljebb akkor fordulhatna elő, amikor $L'[j]$ is részt vesz az összehasonlításban. $L[j]$ azonban minden összehasonlítást megnyert és $L'[j] > L[j]$. Tehát az A algoritmusnak ismét az $L'[i] = L[i]$ elemet kell a legnagyobbnak nyilvánítania, ami ellentmondás.

Például az alábbi közismert program nemcsak legjobb, hanem legrosszabb esetben is $n - 1$ összehasonlítással meghatározza az $L[1 : n]$ lista maximális elemét.

M $(n, L[1 : n], \mathit{legnagyobb})$ *soros eljárás*

Számítási modell: RAM

Bemenet: n (az elemek száma) és $L[1 : n]$ (n hosszúságú,különböző elemeket tartalmazó lista)

Kimenet: *legnagyobb* (a bemeneti lista egyik maximális eleme)

```
legnagyobb := L[1]                ▷ kezdeti érték beállítása
for i := 2 to n
    if L[i] > legnagyobb then
                  legnagyobb := L[i]
                  ▷ legnagyobb frissítése
```

Tehát beláttuk, hogy M az összehasonlítás alapú maximumkeresést a feladat megoldásához okvetlenül szükséges számú összehasonlítással megoldja.

Azokat az algoritmusokat, amelyekre a legjobb és a legrosszabb lépésszám megegyezik, azaz amelyekre

$$W(n) = B(n), \tag{1.64}$$

stabilnak (♣) nevezzük. Megállapításainkat a következőképpen összegezhetjük.

1.14. tétel. (M *abszolút optimális.*) *Ha különböző elemeket tartalmazó, n hosszúságú lista maximális elemét összehasonlítás alapú algoritmussal határozzuk meg, akkor ennek a problémának az időbonyolultsága legjobb, legrosszabb és átlagos esetben is* $n - 1$. *A feladat megoldására* M *abszolút optimális algoritmus.*

Érdemes megemlíteni, hogy a pontos bonyolultságot csak nagyon ritkán tudjuk meghatározni.

1.7.2. Leszámlálás

A leszámlálási módszereket leggyakrabban az átlagos jellemzők meghatározására használjuk. Minden I bemenethez hozzárendelünk egy $\alpha(I)$ jellemző adatot, majd leszámláljuk, hogy mennyi lesz ezen jellemző adatok összege az összes lehetséges bemenetre nézve.

Tekintsük például azt a feladatot, hogy az $1,\ 2,\ \ldots,\ n$ számok különböző permutációit kell rendeznünk úgy, hogy a permutációban szomszédos elemek összehasonlítása (és egyúttal cseréje) a felhasználható művelet. Minden I permutációhoz $\alpha(I)$-ként hozzárendeljük a benne lévő *inverziók* számát és leszámláljuk, hogy összesen hány inverzió van a permutációkban.

1.15. tétel. *Ha $\mathcal{A}$ egy különböző elemekből álló sorozatot a szomszédos elemek összehasonlításával rendező algoritmus, akkor*

$$N(n, \mathcal{A}) \geq \frac{n(n-1)}{4}\ . \tag{1.65}$$

Bizonyítás. Ha $n = 1$, akkor igaz az állítás.

Ha $n \geq 2$, akkor rendeljük minden permutációhoz az *inverzét* , azaz azt a sorozatot, amelyben az elemek fordított sorrendben következnek. Mivel tetszőleges i és j indexre ($1 \leq i,\ \ j \leq n$) igaz, hogy az i-edik és a j-edik elem az egymáshoz rendelt permutációk közül *pontosan* az egyikben van inverzióban, ezért minden párra igaz, hogy bennük együttesen annyi inverzió van, ahány pár (külön-külön), azaz $\binom{n}{2}$. Ezért n elem permutációiban összesen

$$\frac{n!}{2}\binom{n}{2} \tag{1.66}$$

inverzió van. Ha ezt a számot elosztjuk n elem permutációinak számával, megkapjuk a kívánt alsó korlátot. ■

1.7.3. Döntési fák

Egy *döntési fa* (♣) segítségével modellezhetjük az összes döntést, amelyet egy determinisztikus algoritmus végezhet. Egy adott bemenethez tartozó döntések megadnak egy irányított utat a döntési fa gyökerétől valamelyik leveléig, amely megadja az algoritmusnak az adott bemenethez tartozó kimenetét. A döntési fa csak azokat a csúcsokat tartalmazza, amelyekbe legalább egy bemenet esetében eljutunk. A belső csúcsok megfelelnek az algoritmus döntéseinek és az ahhoz tartozó alapműveleteknek (mint például két elem összehasonlítása vagy egy mátrix két elemének összeszorzása). Mivel minden belső csúcshoz legalább egy elvégzendő művelet tartozik, ezért a fa magassága alsó korlát $N(n, A)$-ra.

Hasonlóképpen a levelek *átlagos szintje* (♣) alsó korlát $A(n, \mathcal{A})$-ra.

Bizonyítás nélkül idézünk néhány közismert eredményt, amelyek például döntési fák segítségével igazolhatók.

1.16. tétel. (*Alsó korlát a bináris keresés lépésszámára.*) *Ha* B ´ - - *a rendezett sorozatban binárisan kereső algoritmus, akkor minden pozitív n számra*

$$N(n, \text{B ´ - }) \geq \lceil \lg n \rceil + 1. \tag{1.67}$$

1.17. tétel. (*Alsó korlát az összehasonlítás alapú rendező algoritmusok lépésszámára.*) *Ha* Ö ´ *összehasonlítás alapú rendező algoritmus, akkor*

$$\begin{aligned} N(n, \text{Ö ´}) &\geq \lceil \lg n! \rceil && (1.68)\\ &\geq n \lg n + \frac{\lg n}{2} + O(1). && (1.69) \end{aligned}$$

1.7.4. Tanácsadói érvelés

Ennél a módszernél úgy jutunk alsó korláthoz, hogy dinamikusan olyan bemenetet állítunk elő, amely az algoritmust lehetőleg nagyszámú művelet elvégzésére kényszeríti.

Ez a módszer olyan játékhoz hasonlít, amelyben egy tanácsadónak feltett kérdésekkel juthatunk információhoz, és a tanácsadó arra törekszik, hogy a válaszaival minél kevesebb információt adjon.

A $W(n, \mathcal{A})$ lépésszám alsó becsléséhez úgy jutunk, hogy összeszámoljuk, hány műveletet kellett az adott bemenetre a vizsgált algoritmusnak elvégeznie.

1.7.5. Információelméleti érvelés

Ez a módszer azon alapul, hogy az egy művelettel nyerhető információra felső, a feladat megoldásához szükséges információ mennyiségére pedig alsó korlátot adunk. Ha például egy összehasonlítás lehetséges eredményei **true** és **false**, akkor n összehasonlítással legfeljebb 2^n lehetőséget tudunk megkülönböztetni. Mivel egy n elemet rendező, összehasonlítás alapú algoritmusnak $n!$ lehetőséget kell megkülönböztetnie, ezzel a módszerrel is bizonyíthatjuk a 1.17. tételt.

1.7.6. Gráfelméleti érvelés

Mivel a hálózatokat rendszerint gráfokkal modellezzük, ezért természetes, hogy az alsó korlátok bizonyításában is gyakran szerepelnek gráfok.

Erre a későbbiek során több példát is mutatunk.

1.8. Anomália

A számítógépes rendszerekben *anomáliának* nevezzük azt a jelenséget, amikor egy feladat megoldásához több erőforrást felhasználva rosszabb eredményt kapunk.

Négy konkrét példát említünk. Az egyik a virtuális memória lapjait párhuzamosan használó FIFO (First In – First Out) lapcserélési algoritmussal, a másik a processzorok ütemezésére használt L´ -¨ algoritmussal, a harmadik az átfedéses memóriájú számítógépekben folyó párhuzamos programvégrehajtással, végül a negyedik a P´ - ´ - ´ ´ optimalizációs algoritmussal kapcsolatos.

1.8.1. Lapcsere

Legyenek m, M, n és p pozitív egészek ($1 \leq m \leq M \leq n < \infty$), k nemnegatív egész, $A = \{a_1, a_2, \ldots, a_n\}$ egy véges ábécé. A^k az A feletti, k hosszúságú, A^* pedig az A feletti véges szavak halmaza.

Legyen m egy *kis*, M pedig egy *nagy* számítógép fizikai memóriájában lévő lapkeretek száma, n a háttérmemóriában lévő lapok száma (mindkét számítógépben), A a lapok halmaza.

A lapcserélési algoritmusokat automatákként kezeljük, melyekre m és M a memória mérete, A a bemenő jelek halmaza, $Y = A \cup \{\epsilon\}$ a kimenő jelek halmaza. Ezek az automaták a bemenő jelek $R = (r_1, r_2, \ldots, r_p)$ vagy $R = (r_1, r_2, \ \ldots\ )$ sorozatát dolgozzák fel. Az S_t ($t = 1,\ 2,\ \ldots,$) memóriaállapot a t időpontban (azaz az r_t bemenő jel feldolgozása után) a memóriában tárolt bemenő jelek halmaza. A lapcserélési algoritmusok $S_0 = \{\}$ üres memóriával kezdik a feldolgozást. Egy konkrét P lapcserélési algoritmust a $P = (Q_P, q_0, g_P)$ hármassal definiálunk, ahol Q_P a vezérlő jelek halmaza, $q_0 \in Q_P$ a kezdeti vezérlő jel, g_P az állapot-átmenet függvény, S' az új memóriaállapot, q' az új állapot és y a kimenő jel, S a régi memóriaállapot, q a régi vezérlőállapot és x a bemenő jel.

Az F = FIFO (First In First Out) algoritmus definíciója: $q_0 = ()$ és

$$g_{\mathrm{F}}(S, q, x) = \begin{cases} (S, q, \epsilon), & \text{ha } x \in S, \\ (S \cup \{x\}, q', \epsilon), & \text{ha } x \notin S,\ |S| < m, \\ (S \cap \{y_1\} \cup \{x\}, q”, y_1), & \text{ha } x \notin S \textit{ és } |S| = m, \end{cases} \tag{1.70}$$

ahol $q = (y_1, y_2, \ldots, y_k)$, $q' = (y_1, y_2, \ldots, y_k, x)$ és $q'' = (y_2, y_3, \ldots, y_m, x)$.

A laphibáknak (a memóriaállapot változásainak) a számát $f_P(R, m)$-vel jelöljük. *Anomáliának* (♣) nevezzük azt a jelenséget, amikor $M > m$ és $f_P(R, M) > f_P(R, m)$. Ekkor az $f_P(R, M)/f_P(R, m)$ hányados az *anomália mértéke* (♣).

A P algoritmus hatékonyságát az $E_P(R, m)$ *lapozási sebességgel* (♣) jellemezzük, amit $R = (r_1, r_2, \ldots, r_p)$ véges hivatkozási sorozatra az

$$E_P(R, m) = \frac{f_P(R, m)}{p}, \tag{1.71}$$

$R = (r_1, r_2, \ldots)$ végtelen hivatkozási sorozatra pedig a

$$E_P(R, m) = \liminf_{k \to \infty} \frac{f_P(R_k, m)}{k} \tag{1.72}$$

módon definiálunk, ahol $R_k = (r_1, r_2, \ldots, r_k)$.

Legyen $1 \leq m < n$ és $C = (1, 2, \ldots, n)^*$ egy végtelen ciklikus hivatkozási sorozat. Ekkor $E_{\mathrm{FIFO}}(C, m) = 1$.

Ha végrehajtjuk a $R = (1,2,3,4,1,2,5,1,2,3,4,5)$ hivatkozási sorozatot, akkor $m = 3$ esetében 9, $m = 4$ esetében pedig 10 laphibát kapunk, így $f_{\mathrm{FIFO}}(R, M)/f_{\mathrm{FIFO}}(R, m) = 10/9$.

Bélády, Nelson és Shedler a következő szükséges és elégséges feltételt adták az anomália létezésére.

1.18. tétel. *Akkor és csak akkor létezik olyan hivatkozási sorozat, amelyre a* FIFO *lapcserélési algoritmus anomáliát okoz, ha $m < M < 2m - 1$.*

Az anomália mértékével kapcsolatban pedig a következőt bizonyították.

1.19. tétel. *Ha $m < M < 2m - 1$, akkor tetszőleges $\epsilon > 0$ számhoz létezik olyan $R = (r_1, r_2, \ldots, r_p)$ hivatkozási sorozat, amelyre*

$$\frac{f_{FIFO}(R, M)}{f_{FIFO}(R, m)} > 2 - \epsilon. \tag{1.73}$$

Bélády, Nelson és Shedler a következőt sejtették.

1.20. sejtés. *Tetszőleges R hivatkozási sorozatra és $M > m \geq 1$ memóriaméretekre*

$$\frac{f_{FIFO}(R, M)}{f_{FIFO}(R, m)} \leq 2. \tag{1.74}$$

Ezt a sejtést cáfolja Fornai Péter és Iványi Antal következő tétele, amely szerint az anomália mértéke tetszőlegesen nagy lehet.

1.21. tétel. *Tetszőlegesen nagy L számhoz megadhatók olyan m, M és R paraméterek, melyekre*

$$\frac{f_{FIFO}(R, M)}{f_{FIFO}(R, m)} > L. \tag{1.75}$$

1.8.2. Ütemezés

Tegyük fel, hogy n programot akarunk végrehajtani egy p-processzoros láncon. A végrehajtásnak figyelembe kell vennie a programok közötti megelőzési relációt. A processzorok mohók, és a végrehajtás egy adott L lista szerint történik.

E. G. Coffman jr. 1976-ban leírta, hogy a p processzorszám csökkenése, az egyes programok végrehajtásához szükséges lépések t_i számának csökkenése, a megelőzési korlátozások enyhítése és a lista változtatása külön is anomáliát okozhat.

Legyen a programok végrehajtásának lépésszáma τ, a megelőzési reláció $<$, a lista L és a programok közös listás végrehajtásához szükséges lépések száma p azonos processzorból álló láncon $\omega(p, L, <, \tau)$. Az L' lista, $<' \subseteq <$ megelőzési reláció, $\tau' \leq \tau$ lépésszám vektor és $p' \geq p$ processzorszám esetén a lépésszám legyen $\omega'(p', L', <', \tau')$.

Az anomália mértékét ezúttal a relatív lépésszámmal jellemezzük.

1.22. tétel. (*Ütemezési korlát.*) *Az előbbi feltételek esetében*

$$0 \leq \frac{\omega'}{\omega} \leq 1 + \frac{m-1}{m'}. \tag{1.76}$$

A korlát pontosságát jellemzi a következő állítás.

1.23. tétel. (*Ütemezési korlát élessége.*) *A relatív sebességre adott korlát az m, t, < és L paraméterek mindegyikének változására nézve (külön-külön is) aszimptotikusan éles.*

1.8.3. Párhuzamos feldolgozás átfedéses memóriával

Népszerű formában fogalmazzuk meg az átfedéses memóriájú számítógépek működését modellező párhuzamos algoritmust. A T_0, T_1, $\ldots$ T_r törpék n különböző fajtájú gombócot főznek. Ezeket az egyszerűség kedvéért a $0, 1, \ldots, n$ számokkal jelöljük. Mindegyik törpe végtelen gombócsorozatot állít elő.

Ezeket a gombócokat O_f óriások eszik meg – ahol az f paraméter azt mutatja, hogy az adott óriás az egyes gombócfajtákból legfeljebb hányat ehet meg egy falatban.

Az O_f óriása következőképpen eszik. Első falatához a T_0 törpe sorozatának elejéről a lehető legtöbb gombócot kiválasztja (de egy-egy fajtából legfeljebb f darabot). Ezt a falatot még a T_1, $\ldots$, T_r törpék sorozatának elejéről kiegészíti – az f korlátot betartva.

A további falatokat hasonlóan állítja össze.

Legyen $h_1(f)$, $h_2(f), \ldots$ valószínűségi változók sorozata, melyben a $h_i(f)$ elem az óriás i-edik harapásában lévő gombócok száma. Ekkor az O_f óriás S_f *gombócevési sebességét* (♣) az

$$S_f = \lim_{t\to\infty} M\left(\frac{\sum_{i=1}^{t} h_i}{t}\right) \tag{1.77}$$

határértékkel definiáljuk.

Könnyen belátható, hogy ha $1 \le f \le g$, akkor a gombócevési sebességekre fennáll

$$f \le S_f \le fn, \quad g \le S_g \le gn, \tag{1.78}$$

a két óriás relatív gombócevési sebességére pedig

$$\frac{f}{gn} \le \frac{S_f}{S_g} \le \frac{fn}{g}\ . \tag{1.79}$$

Most megmutatjuk, hogy a *kis óriás* gombócevési sebessége akárhányszor nagyobb lehet, mint a *nagy óriásé*.

1.24. tétel. *Ha $r \geq 1$, $n \geq 3$, $g > f \geq 1$, akkor léteznek olyan gombócsorozatok, amelyekre egyrészt*

$$\frac{S_g}{S_f} = \frac{g}{fn}\,, \tag{1.80}$$

másrészt

$$\frac{S_g}{S_f} = \frac{gn}{f}\,. \tag{1.81}$$

Bizonyítás. A természetes korlátokat megadó (1.79) egyenlőtlenségben szereplő alsó korlát élességének belátásához tekintsük az

$$1^f 2^{2f+1} 1^* \tag{1.82}$$

és

$$1^{f+1}(2\ 3\ \ldots\ n)^* \tag{1.83}$$

sorozatokat.

A felső korlát élességét a

$$1\ 2^{2f+1}\ 1^* \tag{1.84}$$

és

$$1^{f-1}\ 3^{2f}\ 1^f\ (2\ 3\ \ldots\ n)^* \tag{1.85}$$

sorozatok „megevésével” láthatjuk be. ■

Az alsó korlát élessége azt fejezi ki, hogy a kis óriás ehet sokkal kevesebbet – ami természetes. Az n növelésével tetszőlegesen naggyá tehető felső korlát élessége azonban *erős anomália*.

1.8.4. Az anomália elkerülése

Az anomáliát általában igyekeznek elkerülni.

A lapcserélésnél például az elkerülés elégséges feltétele az, ha a cserélési algoritmus rendelkezik a *verem tulajdonsággal* (♣): ha ugyanazt a hivatkozási sorozatot m és $m+1$ méretű memóriájú gépen futtatjuk, akkor minden hivatkozás után igaz az, hogy a nagyobb memória mindazokat a lapokat tartalmazza, amelyeket a kisebb tartalmaz.

A vizsgált ütemezési feladatnál elegendő az, ha nem követeljük meg az ütemező algoritmustól a lista alkalmazását.

1.8.5. Párhuzamos korlátozás és szétválasztás

A *korlátozás és szétválasztás* gyakran alkalmazott optimalizációs módszer.

A módszer alkalmazása során olyan $x = (x_1, x_2, \ldots, x_n)$ vektort keresünk, amely minimalizál egy $f(x)$ függvényt – figyelembe véve korlátozások K halmazát. A korlátozások lehetnek explicitek vagy implicitek. Az implicit korlátozások az x_i értékek kapcsolatát jellemzik, például

$$\sum_{i=1}^{n} a_i x_i \leq b, \tag{1.86}$$

$$a_1 x_1^2 - a_2 x_1 x_2 + a_3 x_3^4 = 6. \tag{1.87}$$

Az explicit korlátozások az x_i értékekre adnak korlátokat, például

$$x_i \in \{0, 1\}, x_i \geq 0. \tag{1.88}$$

Az explicit korlátozásokat kielégítő vektorok alkotják a *megoldási teret*. Ezek a vektorok rendszerint egy fát alkotnak, melyeket *megoldási fának* (♣) nevezünk. A fa gyökerétől bizonyos levelekhez vezető utak a megoldási tér egy elemét határozzák meg. Az ilyen csúcsokat *megengedett megoldás csúcsnak* (♣) nevezzük. Egy ilyen csúcs *költsége* (♣) az f függvény értéke az adott csúcsban. Az optimalizálás célja a *minimális költségű* (♣) csúcs meghatározása.

Az állapotfa minden N csúcsához hozzárendeljük az $f_{min}(N) = \min\{f(Q)|Q$ megengedett megoldás az N csúcs (ha nincs ilyen$Q$, akkor$f_{min}(N) = \infty$). értéket.

A továbbiakban az LCBB (least-cost branch-and-bound) korlátozó-szétválasztó módszerrel foglalkozunk, melyben a következő tulajdonságokkal rendelkező $g()$

heurisztikus függvényt alkalmazzuk:

(T1) $g(N) \leq f_{min}(N)$ az állapottér minden N csúcsára.
(T2) $g(N) = f(N)$ a válaszcsúcsokra.
(T3) $g(N) = \infty$ a megengedett megoldásokra.
(T4) $g(N) \geq g(P)$, ha N a P csúcs gyereke.

$g(\,)$-t *korlátozó függvénynek* (♣) nevezzük. LCBB az állapottérhez tartozó csúcsokat állít elő – $g(\,)$ felhasználásával. Az olyan előállított csúcsot, amelynek gyerekeit még nem állítottuk elő, és megengedett megoldásokhoz vezethet, *élő csúcsnak* nevezzük. Az élő csúcsok listáját karbantartjuk – rendszerint kupacként. LCBB minden iterációs lépésben kiválaszt egy N élő csúcsot, amelynek $g(\,)$ értéke minimális. Ezt a csúcsot aktuális *E-csúcsnak* (♣) nevezzük. Ha N válasz csúcs, akkor minimális költségű válasz csúcs. Ha N nem válasz csúcs, akkor előállítjuk a gyerekeit. Azokat a gyerekeket, amelyek nem vezethetnek minimális költségű válaszhoz, eldobjuk (ezeket a csúcsokat bizonyos heurisztikával választjuk ki). A megmaradó gyerekeket hozzáadjuk az élő csúcsokhoz.

Az LCBB módszer többféleképpen párhuzamosítható. Az egyik lehetőség, hogy minden iterációs lépésben több E-csúcs is kiterjeszthető.

Ha a processzorok száma p, $q = \min\{p, \text{élő csúcsok száma}\}$ csúcsot választunk, mint E-csúcsot (azt a q élő csúcsot, melyekre a legkisebb a $g(\,)$ függvény értéke). Legyen g_{min} ezen csúcsok $g(\,)$ értékeinek minimuma. Ha ezen E-csúcsok bármelyike válaszcsúcs, és a $g(\,)$-értéke g_{min}, akkor az a csúcs minimális költségű válaszcsúcs. Egyébként mind a q csúcsot kiterjesztjük (minden processzor egy csúcsot terjeszt ki), és gyerekeit hozzáadjuk az élő csúcsok listájához. q darab E-csúcs minden ilyen kiterjesztése a párhuzamos LCBB egy iterációja. Adott I-re és g-re jelöljük $I(p)$-vel a p processzor esetében szükséges iterációk számát. Ekkor természetesnek látszanak $I(p)$ következő tulajdonságai:

(I1) ha $p_1 < p_2$, akkor $I(p_1) \geq I(p_2)$;

(I2) $\frac{I(p_1)}{I(p_2)} \leq \frac{p_2}{p_1}$.

(I1) szerint a processzorok számának növelésekor nem nőhet az iterációk száma.

(I2) szerint a teljesítmény nem lehet nagyobb, mint a felhasznált erőforrások mennyisége.

Lai és Sahni 1984-ben megmutatták, hogy egyik tulajdonság sincs biztosítva – még akkor sem, ha $g(\ )$ eleget tesz a (T1)-(T4) korlátozásoknak.

1.9. Gyakorlatok

1-1. Az $\mathcal{A}$ és $\mathcal{B}$ párhuzamos algoritmusok megoldják a kiválasztási feladatot. Az $\mathcal{A}$ algoritmus $n^{0.5}$ processzort használ és a lépésszáma $\Theta(n^{0.5})$. A $\mathcal{B}$ algoritmus n processzort használ és a lépésszáma $\Theta(\lg n)$. Határozzuk meg az elvégzett munkát, a relatív lépésszámot és a hatékonyságot mindkét algoritmusra. Munkahatékonyak-e ezek az algoritmusok?

1-2. Elemezzük a következő két állítást.

a) Az $\mathcal{A}$ algoritmus lépésszáma legalább $O(n^2)$.

b) Mivel az $\mathcal{A}$ algoritmus lépésszáma $O(n^2)$, a $\mathcal{B}$ algoritmus lépésszáma pedig $O(n \lg n)$, ezért a $\mathcal{B}$ algoritmus a hatékonyabb.

1-3. terjesszük ki a váltási hely definícióját nem egész v értékekre és párhuzamos algoritmusokra.

1-4. Becsüljük meg annak valószínűségét, hogy egy szabályos pénzérmét ezerszer feldobva legalább hatszázszor dobunk fejet. *Útmutatás*. A $Pr[X = 600]$ valószínűséget a Stirling-képlettel, a $Pr[X = 601], \ldots, Pr[X = 1000]$ valószínűségeket pedig geometriai sorral becsüljük.

1-5. Egészítsük ki a hálózatok adatait tartalmazó 1.2. táblázatot további adatokkal és hálózatokkal.

1.10. Feladatok

1-1. Variációk O-ra és Ω-ra

Az Ω és O különböző definíciói ismertek. Az egyikre a $\overset{\infty}{\Omega}$ (olvasd: omega végtelen) jelölést használjuk. Azt mondjuk, hogy $f(n) = \overset{\infty}{\Omega}(g(n))$, ha létezik c pozitív valós állandó úgy, hogy $f(n) \geq cg(n) \geq 0$ teljesül végtelen sok egész n-re.

a. Egy függvényt aszimptotikusan nemnegatívnak nevezünk, ha minden elég nagy n-re nemnegatív. Mutassuk meg, hogy bármely két aszimptotikusan nemnegatív $f(n)$ és $g(n)$ függvény esetében vagy $f(n) = O(g(n))$, vagy $f(n) = \overset{\infty}{\Omega}(g(n))$, vagy mindkettő teljesül, ha azonban $\overset{\infty}{\Omega}$ helyett Ω-t használunk, akkor nem igaz az állítás.

b. Mik a lehetséges előnyei és hátrányai annak, ha a programok futási idejének jellemzésére $\overset{\infty}{\Omega}$-t használunk Ω helyett?

Néhány szerző a O-t is kissé másképp definiálja; használjuk az O' jelölést erre az alternatív definícióra. Azt mondjuk, hogy $f(n) = O'(g(n))$ akkor és csak akkor, ha $|f(n)| = O(g(n))$.

c. Ismert, hogy $f(n) = O(g(n))$ akkor és csak akkor teljesül, ha $f(n) = \Omega(g(n))$ és $f(n) = O(g(n))$. Mit mondhatunk, ha Ω helyett $\overset{\infty}{\Omega}$ szerepel?

Szokásos a $\tilde{O}$-n (olvasd: gyenge ordó) szimbólumot a logaritmikus tényezők elhanyagolásával kapott O-jelölésre használni. Azt mondjuk, hogy $\tilde{O}(g(n)) = f(n)$, ha léteznek olyan c pozitív valós, k és n_0 pozitív egész állandók úgy, hogy, ha $n \geq n_0$, akkor $0 \leq f(n) \leq cg(n)\lg^k(n)$.

d. Definiáljuk hasonló módon $\tilde{\Omega}$-t és $\tilde{\Theta}$-t. Bizonyítsuk be a **c.** részben kimondott állítás ennek megfelelő változatát.

1-2. A relatív lépésszám korlátai

a. Bizonyítsuk be Amdahl és Gustafson törvényét.

b. Magyarázzuk meg, hogy a két törvény közötti ellentmondás látszólagos.

c. A gyakorlatban milyen korlátai vannak a relatív lépésszámnak?

1-3. Hanoi tornyai általánosan

Általánosítsuk a Hanoi tornyaira vonatkozóan kapott eredményt.

a. Mit mondhatunk a lépésszámról akkor, ha négy rudat használhatunk?

b. Mit mondhatunk a lépésszámról akkor, ha $2 + k$ ($k \geq 1$) rudat használhatunk?

c. Mit mondhatunk a lépésszámról akkor, ha korlátozzuk a rudak közötti mozgatás lehetőségeit, például az A rúdról a B rúdra és a B rúdról az A rúdra nem szabad korongot áthelyezni?

d. Mit mondhatunk a lépésszámról akkor, ha s szerzetes párhuzamosan dolgozhat? Feltételezzük, hogy minden lépésben minden rúdról legfeljebb egy korongot szabad elvenni és minden rúdra legfeljebb egy korongot szabad ráhelyezni, továbbá minden szerzetes legfeljebb egy korongot mozgathat?

e. Mit mondhatunk a lépésszámról akkor, ha az s szerzetes párhuzamos munkáját paraméteresen értelmezzük: az m-párhuzamos olvasás jelentse azt, hogy a szerzetesek egy lépésben rudanként legfeljebb az m legfelső koronghoz férnek hozzá; az m-párhuzamos írás pedig jelentse azt, hogy minden korongra külön legfeljebb m korongot helyezhetnek egyidejűleg.

1-4. Anomália

Tervezzünk egy algoritmust (és valósítsuk is meg), amely azt bizonyítja, hogy egy adott feladatot $q > p$ processzoron megoldani tovább tart, mint $p > 1$ processzoron.

1-5. Párhuzamossággal a hírnévért és dicsőségért

1997-ben Andrew Beale dallasi bankár 50 ezer dollár díjat tűzött ki annak, aki bizonyítja vagy cáfolja sejtését. Eszerint ha

$$a^q + b^p = c^r, \tag{1.89}$$

akkor az a, b és c számoknak van egynél nagyobb közös osztója (az egyenletben mind a hat betű egész számot jelent és a kitevők értéke legalább három).

Hogyan használhatók fel a párhuzamos algoritmusok a díj megszerzésére?

1-6. Rangsorolás

Tekintsük az összehasonlítás alapú rendezés következő általánosítását. n elemet – például n sportolót – páronként összehasonlítunk, és a *győztesnek* 1 pontot, a vesztesnek pedig 0 pontot adunk. Legyen p_i az i-edik játékos *pontszáma* (győzelmeinek száma).

a. Tervezzünk párhuzamos algoritmust, amely adott $\mathbf{q}= q_1, q_2, \ldots, q_n$ sorozatról eldönti, hogy lehet-e az előbb leírt *verseny* pontsorozata. *Útmutatás.* Használjuk fel Landau tételét, amely szerint $\mathbf{q}$ akkor és csak akkor lehet pontsorozat, ha a következő két feltétel mindegyike teljesül:

$$\sum_{i=1}^{k} q_i \geq \binom{k}{2} \quad (\text{ha } 1 \leq k \leq n) \tag{1.90}$$

és

$$\sum_{i=1}^{n} q_i = \binom{n}{2}. \tag{1.91}$$

b. Oldjuk meg a feladatot akkor, ha minden összehasonlításnál $k \geq 1$ pontot osztunk ki az összehasonlított sportolók között, és a k pont minden lehetséges módon felosztható (azzal a megszorítással, hogy mindkét játékosra a kapott pontok száma nemnegatív egész).

b. Oldjuk meg a feladatot akkor, ha minden összehasonlításnál k ($1 \leq a \leq k \leq b$) pontot osztunk ki az összehasonlított sportolók között, – a k pont minden lehetséges módon felosztható (azzal a megszorítással, hogy mindkét játékosra a kapott pontok száma nemnegatív egész), az a és b számok pozitív egészek.

2. Párhuzamos gépek

2.1. Alapvető módszerek

Ebben az alfejezetben két, a feladatok párhuzamos megoldásában gyakran használt módszert mutatunk be. Az asszociatív műveletek gyors elvégzését a prefixek számítására, a mutatóugrást pedig a listarangsorolási feladat megoldására használjuk fel.

2.1.1. Prefixszámítás

Legyen Σ egy alaphalmaz, melyen definiáltuk a $\oplus$ *bináris asszociatív operátort* (♣). Feltesszük, hogy a művelet egy lépéssel elvégezhető és a Σ halmaz zárt erre a műveletre nézve.

Egy $\oplus$ bináris operátor *asszociatív* (♣) a Σ alaphalmazon, ha minden $x, y, z \in \Sigma$ esetében teljesül

$$((x \oplus y) \oplus z) = (x \oplus (y \oplus z)). \tag{2.1}$$

Legyenek az $X = x_1, x_2, \ldots, x_p$ sorozat elemei a Σ alaphalmaz elemei. Ekkor a prefixszámítás bemenő adatai az X sorozat elemei, a *prefixszámítási feladat* (♣) pedig az $x_1,\ x_1 \oplus x_2,\ \ldots, x_1 \oplus x_2 \oplus x_3 \oplus \ldots \oplus x_p$ elemek (♣) meghatározása. Ezeket a meghatározandó elemeket *prefixeknek* hívjuk.

Érdemes megjegyezni, hogy más területeken inkább az X sorozat $x_1, x_2, \ldots, x_k$ kezdősorozatait hívják prefixeknek.

2.1. példa. Asszociatív műveletek. Ha Σ az egészek halmaza, $\oplus$ az összeadás és a bemenő adatok sorozata a 3, -5, 8, 2, 5, 4, akkor a prefixek 3, -2, 6, 8, 13, 17. Ha az ábécé és a bemenő adatok ugyanazok, de a művelet a szorzás, akkor a kimenő adatok (prefixek) $3, -15, -120, -240, -1200, -4800$. Ha a művelet a minimum (ez is asszociatív), akkor a prefixek $3, -5, -5, -5, -5, -5$. Az utolsó prefix megegyezik a bemenő számok legkisebbikével. ♠

A prefixszámítás sorosan megoldható $O(p)$ lépéssel. Bármely $\mathcal{A}$ soros algoritmusnak $N(p, \mathcal{A}) = \Omega(n)$ lépésre szüksége van. Vannak olyan párhuzamos algoritmusok, amelyek különböző párhuzamos modelleken megoldják a munkahatékony prefixszámítást.

Először a CREW- CREW PRAM algoritmust mutatjuk be, amely p processzoron $\Theta(\lg p)$ lépéssel számítja ki a prefixeket.

Azután az EREW- algoritmus következik, amelynek menynyiségi jellemzői hasonlóak az előző algoritmuséhoz, azonban EREW PRAM számítási modellen is végrehajtható.

Ezekkel az algoritmusokkal a prefixszámítást a soros megoldás $\Theta(p)$ lépésszámánál lényegesen gyorsabban meg tudjuk oldani, de sok az elvégzett munka.

Ezért is érdekes a H ´ -P CREW PRAM algoritmus, amely $\left\lceil \frac{p}{\lg p} \right\rceil$ processzort használ és $O(\lg p)$ lépést tesz. Ennek elvégzett munkája csak $O(p)$, ezért a hatékonysága $\Theta(1)$ és az algoritmus munkahatékony. Ennek az algoritmusnak a gyorsítása $\Theta(\frac{n}{\lg n})$.

A továbbiakban – a jelölések egyszerűsítése érdekében – a $\left\lceil \frac{p}{\lg p} \right\rceil$ típusú kifejezések helyett általában csak $\frac{p}{\lg p}$-t írunk.

Az algoritmusok tervezéséhez az *oszd meg és uralkodj* elvet alkalmazzuk. A bemenő adatok legyenek $X = x_1,\ x_2,\ \ldots,\ x_p$. Az általánosság megszorítása nélkül feltehető, hogy p a 2 egész kitevőjű hatványa.

2.1.1.1. Prefixszámítás CREW PRAM modellen

Először egy p-processzoros algoritmust mutatunk be, melynek lépésszáma $\Theta(\lg p)$.

CREW- (p, X, Y) *párhuzamos rekurzív eljárás*

Számítási modell: PRAM

Bemenet: p (a bemenő sorozat hossza) és
$X[1:p] = x_1,\ x_2, \ldots, x_p$ (p hosszúságú sorozat)

Kimenet: $Y[1:p] = y_1,\ y_2,\ \ldots,\ y_p$ (n hosszúságú sorozat, amelynek elemei a prefixek)

if $p = 1$ **then**
  $y_1 := x_1$
if $p > 1$ **then**
  Az első $\frac{p}{2}$ processzor rekurzívan kiszámítja az $x_1, x_2, \ldots, x_{p/2}$-höz tartozó prefixeket – legyenek ezek $y_1, y_2, \ldots, y_{p/2}$ (ezek adják az végeredmény első felét). Ugyanakkor a többi processzor rekurzívan kiszámítja az $x_{p/2+1}, x_{p/2+2}, \ldots, x_p$-hez tartozó prefixeket – legyenek ezek $y_{p/2+1}, y_{p/2+2}, \ldots, y_p$.
  A processzorok másik fele párhuzamosan kiolvassa a globális memóriából $y_{p/2}$-t és az $y_{p/2} \oplus y_{p/2+1}, y_{p/2} \oplus y_{p/2+2}, \ldots, y_{p/2} \oplus y_p$ prefixeket számítva előállítja a végeredmény másik felét.

2.2. példa. 8 elem prefixeinek számítása 8 processzoron

Legyen $n = 8$ és $p = 8$. A prefixszámítás bemenő adatai 12, 3, 6, 8, 11, 4, 5 és 7, az asszociatív művelet az összeadás. Az első szakaszban az első 4 processzor 12, 3, 6, 8 bemenethez a 12, 15, 21, 29 prefixeket számolja ki. A másik 4 processzor pedig a 11, 4, 5, 7 bemenethez számolja ki a 11, 15, 20, 27 prefixeket.

A második szakaszban az első 4 processzor nem dolgozik, a második 4 pedig 29-et ad minden prefixhez és a 40, 44, 49, 56 eredményt kapja. ♠

Mi ennek az algoritmusnak a $T(p)$ lépésigénye? Az első lépés $T(\frac{p}{2})$ ideig, a második pedig $O(1)$ ideig tart. Ezért a következő rekurziót kapjuk:

$$T(p) = T\left(\frac{p}{2}\right) + O(1), \tag{2.2}$$

$$T(1) = 1. \tag{2.3}$$

Ennek a rekurzív egyenletnek a megoldása $T(p) = O(\lg p)$.

2.3. tétel. *A* CREW- *algoritmus p* CREW PRAM *processzoron* $\Theta(\log p)$ *lépésben számítja ki p elem prefixeit.*

Ez az algoritmus nem munkahatékony, mivel $\Theta(p \lg p)$ munkát végez, és ismert olyan soros algoritmus, amely $O(p)$ lépést tesz. Munkahatékony algoritmust kaphatunk például úgy, ha a felhasznált processzorok számát lecsökkentjük $\frac{p}{\lg p}$-re, miközben a lépésszám nagyságrendje ugyanaz marad. A processzorszámot úgy csökkentjük, hogy a bemenet méretét csökkentjük $\frac{p}{\lg p}$-re, alkalmazzuk az előző algoritmust, majd végül minden prefixet kiszámolunk.

2.1.1.2. Prefixszámítás EREW PRAM modellen

A következő algoritmusban a párhuzamos írás helyett elég a soros írás lehetősége.

EREW- (p, X, Y) *párhuzamos rekurzív eljárás*

Számítási modell: EREW PRAM

Bemenet: p (a bemenő sorozat hossza) és $X[0:p] = x_0,\ x_1,\ x_2,$
$\ldots,\ x_p$ (p hosszúságú sorozat)

Kimenet: $Y[1:p] = y_1,\ y_2,\ \ldots,\ y_p$ (p hosszúságú sorozat, melynek elemei a prefixek)

01 P_i **in parallel for** $(i = 1, 2, \ldots, p$
02 $\qquad Y[i] := X[i-1] \oplus X[i]$
03 $k := 2$
04 **while** $k < p$
05 $\qquad P_i$ **in parallel for**
06 $\qquad\qquad Y[i] := Y[i-k] \oplus Y[i]$
07 $\qquad k := k + k$

2.4. tétel. *Az* EREW- *algoritmus p* EREW PRAM *processzoron* Θlog *p lépésben számítja ki p elem prefixeit.*

Bizonyítás. A lépésszám nagyságrendjét az határozza meg, hányszor hajtódik végre a hetedik utasítás. ■

2.1.1.3. Prefixszámítás munkahatékonyan

H ´ - (p, X, Y) *párhuzamos rekurzív eljárás*

Számítási modell: CREW PRAM

Bemenet: p (a bemenő sorozat hossza) és $X[1:p] = x_1,\ x_2, \ldots,\ x_p$

(p hosszúságú sorozat)

Kimenet: $Y[1:p] = y_1, y_2, \ldots, y_p$ (p hosszúságú sorozat, melynek elemei a prefixek)

01 P_i processzor ($i = 1, 2, \ldots, \frac{p}{\lg p}$) rekurzívan számolja a hozzárendelt $\lg p$ darab $x_{(i-1)\lg p+1}, x_{(i-1)\lg p+2}, \ldots, x_{i \lg p}$ elem prefixeit. Legyen az eredmény $z_{(i-1)\lg p+1}, z_{(i-1)\lg p+2}, \ldots, z_{i \lg p}$.

02 Összesen $\frac{p}{\lg p}$ processzor együtt alkalmazza a CREW-algoritmust a $\frac{p}{\lg p}$ darab elem, $z_{\lg p}, z_{2\lg p}, z_{3\lg p}, \ldots, z_p$ prefixeinek számítására. Legyen az eredmény $w_{\lg p}, w_{2\lg p}, w_{3\lg p}, \ldots, w_p$.

03 Minden processzor aktualizálja az első lépésben kiszámolt értéket: a P_i processzor ($i = 2, 3, \ldots, \frac{p}{\lg p}$) kiszámolja a $w_{(i-1)\lg p} \oplus z_{(i-1)\lg p+1}, w_{(i-1)\lg p} \oplus z_{(i-1)\lg p+2}, \ldots, w_{(i-1)\lg p} \oplus z_{i \lg p}$ prefixeket, majd az első processzor változtatás nélkül kiadja a $z_1, z_2, \ldots, z_{\lg p}$ prefixeket.

Az algoritmus lépésszáma logaritmikus. Ennek belátását megkönnyíti a következő két képlet:

$$z_{(i-1)\lg p+k} = \sum_{j=(i-1)\lg p+1}^{i \lg p} x_j \quad (k = 1,\ 2,\ \ldots,\ \lg p) \tag{2.4}$$

és

$$w_{i \lg p} = \sum_{j=1}^{i} z_{j \lg p} \quad (i = 1,\ 2,\ \ldots,), \tag{2.5}$$

ahol az összegzés a megfelelő asszociatív művelet segítségével történik.

2.5. tétel. (*Párhuzamos prefixszámítás $\Theta(\lg p)$ lépéssel.*) H ´ -
$\frac{p}{\lg p}$ CREW PRAM *processzoron $\Theta(\lg p)$ lépéssel számítja ki p elem prefixeit.*

Bizonyítás. Az algoritmus az első szakasza $O(\lg p)$ ideig, a második szakasza $O(\lg \frac{p}{\lg p}) = O(\lg p)$ lépésig tart. Végül a harmadik szakasz ugyancsak $O(\lg p)$ lépést igényel.

■

A tételből következik, hogy a H ´ - algoritmus munkahatékony.

2.6. példa. 16 elem összeadása

Legyen 16 elemünk: 5, 12, 8, 6, 3, 9, 11, 12, 1, 5, 6, 7, 10, 4, 3, 5. Az asszociatív művelet az összeadás. Ekkor $\lg n = 4$. Ekkor az első párhuzamos lépésben a processzorok 4-4 elem prefixeit számolják. A második lépésben a helyi összegekből globális összegeket számolunk, majd azokkal a harmadik lépésben frissítjük a helyi eredményeket. A számítás menetét mutatja a 2.1. ábra. ♠

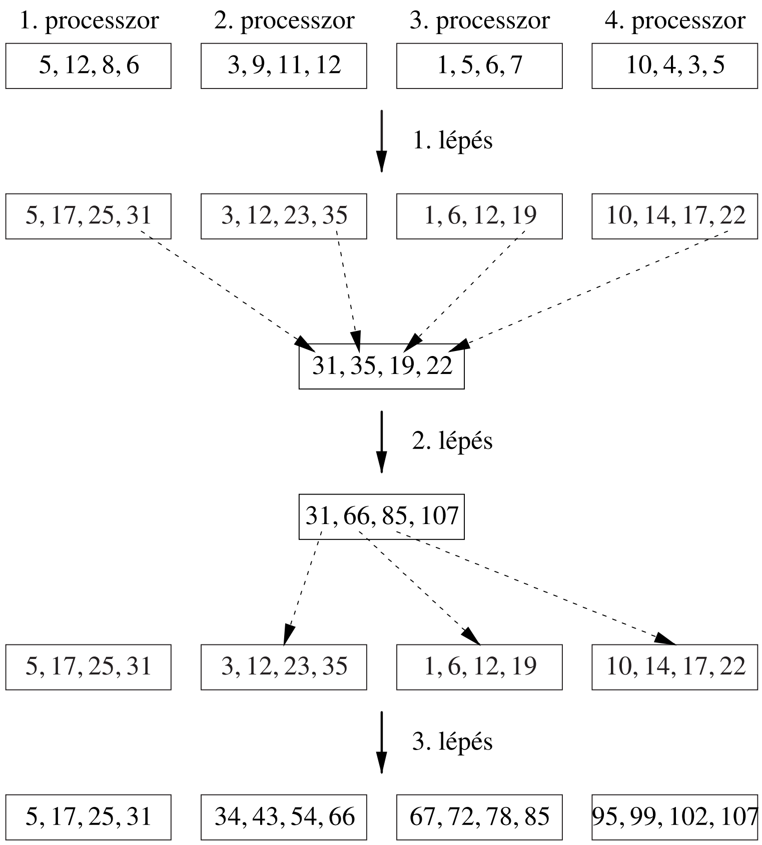
2.1.2. Tömb elemeinek rangsorolása

A *tömbrangsorolási probléma* (♣) bemenő adata egy p elemű tömbben ábrázolt lista: minden elem tartalmazza jobboldali szomszédjának az indexét (és esetleges további adatokat). A feladat az elemek *rangjának* (♣) (jobboldali szomszédai számának) meghatározása.

Mivel az adatokra nincs szükség a megoldáshoz, feltesszük, hogy az elemek csak a szomszéd indexét tartalmazzák. A jobbszélső elem index mezője nulla. Az indexet a továbbiakban mutatónak hívjuk.

2.7. példa. Tömbrangsorolás bemenő adatai

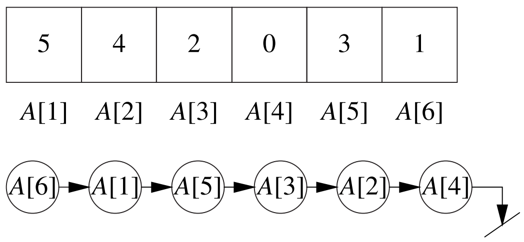
Legyen $A[1 : 6]$ a 2.2. ábra felső sorában bemutatott tömb. Ekkor az $A[1]$ elem jobboldali szomszédja $A[5]$, $A[2]$ jobboldali szomszédja $A[4]$. $A[4]$ az utolsó elem, ezért rangja 0. $A[2]$ rangja 1, mivel csak $A[4]$ van tőle jobbra. $A[5]$ rangja 3, mivel az $A[3], A[2]$ és $A[4]$ elemek vannak tőle jobbra. Az elemek sorrendjét



2.1. ábra. 16 elem prefixeinek számítása a H ´ - algoritmussal

(balról jobbra haladva) mutatja az ábra alsó része. ♠

A tömbrangsorolás sorosan elvégezhető lineáris lépésszámmal. Először meghatározzuk a *tömb fejét* (♣) – az egyetlen olyan i értéket ($1 \leq i \leq p$), melyre $A[j] \neq i$ teljesül minden $1 \leq j \leq n$ értékre. Legyen $A[i]$ a tömb feje. A fejtől kiindulva pásztázzuk a tömböt és az elemekhez rendre hozzárendeljük a $p, p-1, \ldots, 1, 0$ rangokat.



2.2. ábra. Tömbrangsorolási probléma bemenő adatai és a megfelelő tömb

Ebben a részben két párhuzamos algoritmust ismertetünk. Az egyik a D - , amely egy p-processzoros EREW PRAM $\overline{O}(\lg p)$ lépésszámmal, a másik a V´ - , amely egy $\frac{p}{\lg p}$-processzoros véletlenített EREW PRAM algoritmus $\overline{O}(\lg p)$ lépésszámmal. Mindkettő relatív sebessége $\Theta(\frac{p}{\lg n})$. Az első algoritmus hatékonysága $\frac{\Theta(p)}{\Theta(p \lg p)} = \Theta(\frac{1}{\lg p})$, míg a másodiké $\Theta(1)$. Ezért a második algoritmus munkahatékony, az első viszont nem az.

2.1.2.1. Determinisztikus tömbrangsorolás

Ezekben az algoritmusokban az egyik alapvető ötlet a *mutatóugrás*. D - szerint először mindegyik elem a jobboldali szomszédjának indexét tartalmazza, és ennek megfelelően a rangja – a jobboldali szomszédjához viszonyítva – 1 (kivétel a lista utolsó eleme, melynek rangja 0. Ezt a kezdeti állapotot mutatja a 2.3. ábra első sora.

Ezután módosítjuk a csúcsokat úgy, hogy mindegyik a jobboldali szomszédjának a jobboldali szomszédjára mutasson (ha nincs, akkor a lista végére). Ezt tükrözi a 2.3. ábra második sora.

Ha p processzorunk van, akkor ez $O(1)$ lépéssel elvégezhető.

Most minden csúcs (kivéve az utolsót) olyan csúcsra mutat, amelyik eredeti-

szomsz						*rang*						
5	4	2	0	3	1	1	1	1	0	1	1	(kezdeti állapot)
3	0	4	0	2	5	2	1	2	0	2	2	$q = 1$
4	0	0	0	0	2	4	1	2	0	3	4	$q = 2$
0	0	0	0	0	0	4	1	2	0	3	5	$q = 3$

2.3. ábra. A D - algoritmus működése a 2.6. példa adataival

leg 2 távolságra volt. A mutatóugrás következő lépésében a csúcsok olyan csúcsra mutatnak, amelyek eredetileg 4 távolságra voltak tőlük (ha ilyen csúcs nincs, akkor a lista végére) – amint azt az ábra harmadik sora mutatja.

A következő lépésben a csúcsok (pontosabban a mutató részük) a 8 távolságú szomszédra mutatnak (ha van ilyen – ha nincs, akkor a lista végére), az 2.3. ábra utolsó sora szerint.

Minden csúcs minden lépésben információt gyűjt arról, hány csúcs van közte és azon csúcs között, amelyre most mutat. Ehhez kezdetben legyen a csúcsok rang mezőjében 1 – kivéve a jobboldali csúcsot, melyre ez az érték legyen 0. Legyen $rang[i]$ és $szomsz[i]$ az i csúcs rang, illetve szomszéd mezője. A mutatóugrás során $rang[i]$-t általában $rang[i] + rang[szomsz[i]]$-re módosítjuk – kivéve azokat a csúcsokat, melyekre $szomsz[i] = 0$. Ezután $szomsz[i]$-t úgy módosítjuk, hogy $szomsz[szomsz[i]]$-re mutasson. A teljes D - algoritmus a következő.

D - $(szomsz[1 : p], rang[1 : p])$ *párhuzamos eljárás*

Számítási modell: EREW PRAM

Bemenet: $szomsz[1 : p]$ (az elemek jobboldali szomszédainak indexei)

Kimenet: *rang*$[1 : p]$ (az elemek rangjai)

P_i **in parallel for** $i := 1$ **to** n
        if $szomsz[i] = 0$ **then**
                $rang[i] := 0$
            else
                $rang[i] := 1$
for $i := 1$ **to** $\lceil \lg n \rceil$
    P_i **in parallel for** $1 \leq i \leq n$
        if $szomsz[i] \neq 0$ **then**
            $rang[i] := rang[i] + rang[szomsz[i]]$
            $szomsz[i] := szomsz[szomsz[i]]$

A 2.3. ábra mutatja, hogyan működik D - a 2.6. példa adataival.

Kezdetben minden csúcs rangja 1, kivéve a 4. csúcsot. Amikor $q = 1$, akkor például az 1. csúcs *rang* mezőjét kettőre változtatjuk, mert jobboldali szomszédjának (ez az 5. csúcs) rangja 1. Az 1. csúcs *szomsz* mezőjét az 5. csúcs szomszédjának indexére, azaz 3-ra változtatjuk.

2.8. tétel. *A* D - *algoritmus egy* EREW PRAM *modellen* $\Theta(\lg p)$ *lépésben határozza meg egy p elemű tömb elemeinek rangját.*

Mivel a A D - algoritmus $p \lg p$ munkát végez, ezért nem munkahatékony.

A listarangsorolási probléma megfelel a lista prefix összege számításának, ahol minden csúcs súlya 1, kivéve a jobboldalit, melynek súlya 0. A D - algoritmus könnyen módosítható úgy, hogy kiszámítsa egy lista prefixeit – a processzorszámra és a lépésszámra vonatkozó hasonló korlátokkal.

2.1.2.2. Véletlenített listarangsorolás (⋆)

Most egy munkahatékony véletlen listarendező algoritmus következik. Minden processzor $\lg p$ csúcs rangját számolja. A P_i processzort az $A[(i-1)\lg p+1], A[(i-1)\lg p+2], \ldots, A[i \lg p]$ csúcsokhoz rendeljük. Az algoritmus meneteket hajt végre. Minden menetben kiválasztja és *kiemeli* (♣) a csúcsok egy részét. Amikor egy csúcsot kiemelünk, akkor a rá vonatkozó információt tároljuk úgy, hogy később a rangját meg tudjuk határozni. Ha már csak 2 csúcs marad, a listarendezési probléma egyszerűen megoldható. A következő menetben a kiemelt csúcsokat *beemeljük*.

Amikor egy csúcsot *beemelünk* (♣), akkor a helyes rangját is meghatározzuk. A beemelés sorrendje a kiemelési sorrend fordítottja. A K algoritmus a következő.

K (p) *párhuzamos eljárás*

Számítási modell: EREW PRAM

Bemenet: p (a rangsorolandó elemek száma) és $A[1:p]$
(a rangsorolandó elemeket tartalmazó tömb)

Kimenet: $rang[1:p]$ (az elemek rangjai)

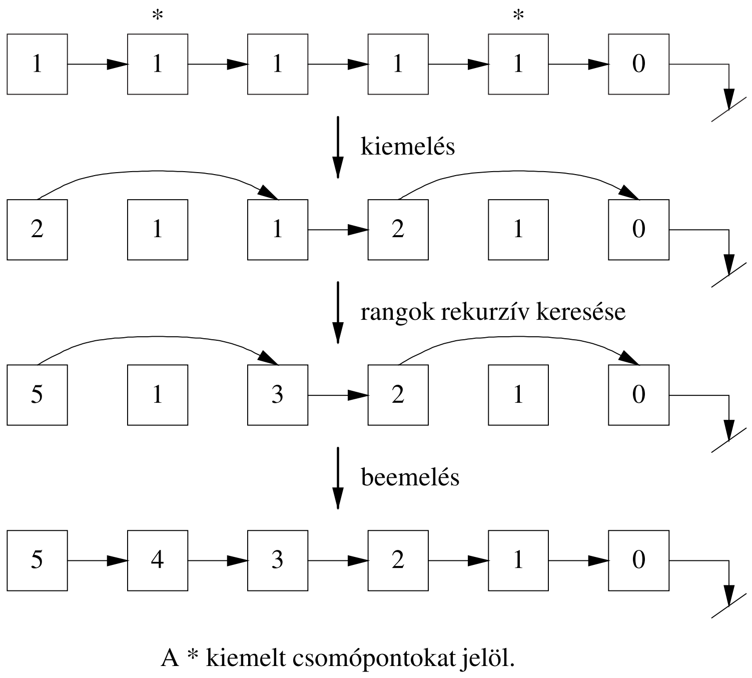
Láncoljuk a listát kettősen. Legyen az $A[i]$ csúcs bal oldali
  szomszédja *bal_szomsz*$[i]$, jobboldali szomszédja
  jobb_szomsz$[i]$ csúcsok rang mezőit kezdetben így adjuk meg:
  $[1] \to [1] \to [1] \to [1] \to [1] \to [1] \to [0] \Downarrow$.
while A megmaradó csúcsok száma legalább 3
    P_i **in parallel for** $1 \le i \le \frac{p}{\lg p}$
      megvizsgálja a következő hozzátartozó kiemeletlen
      csúcsot (legyen ez x). Ezután feldob egy kétoldalú
      érmét. Ha az eredmény *írás*, akkor P_i tétlen

marad a menet hátralévő részében. A következő menetben ismét megpróbálja x-et kiemelni. Másrészt, ha a dobás eredménye *fej*, akkor P_i megvizsgálja, vajon a jobboldali szomszédját vizsgálta-e a megfelelő processzor. Ha a megfelelő processzor vizsgálta és a dobásának eredménye ugyancsak *fej*, akkor P_i feladja és a fokozat hátralévő részében tétlen marad. Ha nem, akkor P_i kiemeli x-et.

04 Amikor egy x csúcsot kiemelünk, tároljuk a fokozatszámot, valamint a *bal_szomszéd* mutatót és $rang[bal\_szomsz[x]]$-et. Az utóbbi ebben a pillanatban a *bal_szomsz* és x közötti csúcsok számát tartalmazza. P_i a $rang[bal\_szomsz[x]] := rang[bal\_szomsz[x]] + rang[x]$ értékadást is elvégzi. P_i végül beállítja a $jobb\_szomsz[bal\_szomsz[x]] := jobb\_szomsz[x]$ és $bal\_szomsz[jobb\_szomsz[x]] := bal\_szomsz[x]$ értékeket.

A kettős láncolás p processzorral $O(1)$ lépéssel elvégezhető. A P_i processzorhoz az $A[i]$ $(1 \le i \le p)$ csúcsokat rendeljük. Egy lépésben P_i a $szomsz[i]$ memóriarekeszbe ír úgy, hogy a következő lépésben az $A[szomsz[i]]$ csúccsal összekapcsolt processzor ismerni fogja baloldali szomszédját. A lassulási lemma segítségével belátható, hogy $\frac{n}{\lg n}$ processzoron p elem kettős láncolása $\lg p$ lépésben elvégezhető.

A csúcsok beemelése (lásd 2.4. ábra) ugyancsak fokozatokban történik. Amikor az x csúcsot beemeljük, helyes rangját így határozzuk meg: ha kiemelésekor a $bal\_szomsz[x]$ mutatót tároltuk, akkor x rangját úgy kapjuk, hogy $bal\_szomsz[x]$ aktuális rangjából levonjuk azt a rangot, amit x kiemelésekor tároltunk. A mutatókat ugyancsak aktualizáljuk, figyelembe véve azt a tényt, hogy x-et már beemeltük. Ezt mutatja a **??**. ábra, amelyen a csúcsoknak csak a jobboldali mutatója szerepel.



2.4. ábra. A csúcsok kiemelése és beemelése

Most megmutatjuk, hogy a kiemelések s összes száma $\overline{O}(\lg p)$. Ha egy csúcsot a i-edik fokozatban kiemelünk, akkor a $(2s - i + 1)$. fokozatban fogjuk beemelni. Így az algoritmus szerkezete a következő. Az $1, 2, \ldots, s$ fokozatokban egymás után kiemeljük a csúcsokat. Az s fokozatban az addigra megmaradt 2 csúcs egyikét kiemeljük. Az $(s + 1)$-edik fokozatban beemeljük az s fokozatban kiemelt csúcsot. Az $(s + 2)$-edik fokozatban az $s = 1$ fokozatban kiemelt csúcsot emeljük be és így tovább. Az utolsó fokozat után az eredeti lista minden csúcsának rangját tudjuk.

Mivel P_i minden fokozatban csak egy csúcsot vizsgál, ezért a hozzá tartozó csúcsok közül legfeljebb egyet emelünk ki. Az is minden esetben fennáll, hogy a lista szomszédos csúcsait egyszerre nem emeljük ki: ugyanis a *fej* után egy processzor csak akkor próbálja kiemelni a választott csúcsot, ha a jobboldali szomszéd processzor nem fej-et dobott. Ezért a processzorok fokozatonként csak $O(1)$ lépést igényelnek.

Az algoritmus lépésszámához csak s-et kell meghatározni. Ehhez megbecsüljük a fokozatonként kiemelt csúcsok számát. Minden P_i processzorra igaz, hogy a kiválasztott x csúcsot legalább $\frac{1}{4}$ valószínűséggel kiemeli: ugyanis P_i $\frac{1}{2}$ valószínűséggel dob fej-et, és legalább $\frac{1}{2}$ annak a valószínűsége, hogy x jobboldali szomszédját (legyen ez a csúcs y) nem választjuk ki, vagy kiválasztjuk ugyan, de y processzora írást dob.

Minden processzor $\lg p$ csúccsal kezdi az algoritmust és minden fokozatban legalább $\frac{1}{4}$ annak valószínűsége, hogy kiemelünk egy csúcsot. Ezért s várható értéke legfeljebb $4 \lg p$. Most alkalmazzuk az (1.51) Csernov-egyenlőtlenséget, amely p darab q-paraméterű Bernoulli-kísérlet esetében minden $0 < \epsilon < 1$ számra igaz. A $12\alpha \lg p$ és $\frac{1}{2}$ paraméterekkel az $\epsilon = \frac{2}{3}$ esetben azt kapjuk, hogy

$$P(s \leq 12\alpha \lg p) > 1 - p^{-\alpha} \tag{2.6}$$

minden $\alpha \geq 1$ értékre. Tehát beláttuk a következő tételt.

2.9. tétel. (*Lista rendezése* $\overline{O}(\lg p)$ *lépéssel.*) *A* K *algoritmus egy p hosszúságú lista rangsorolását* $O\left(\frac{p}{\lg p}\right)$ EREW PRAM *processzoron* $\overline{O}(\lg p)$ *lépéssel elvégezi.*

2.2. Kiválasztás

Adott $n \geq 2$ kulcs és egy i $(1 \leq i \leq n)$ egész szám. A feladat az i-edik legkisebb kulcs kiválasztása. Mivel a kiválasztáshoz minden elemet meg kell vizsgálni, ezért $N(n) = \Omega(n)$. Erre a feladatra ismert olyan $\mathcal{A}$ soros algoritmus, amelyikre $W(n, \mathcal{A}) = O(n)$, tehát $\mathcal{A}$ aszimptotikusan optimális.

Ehhez hasonló a *keresési feladat*, amelyben azt kell eldönteni, hogy adott elem előfordul-e a vizsgált sorozatban – és ha igen, milyen indexszel. Ennél a feladatnál tagadó válasz is lehetséges és egy elemről tulajdonságai alapján eldönthető, megfelel-e a keresési feladatnak.

Először 3 speciális esetet vizsgálunk, majd egy munkahatékony véletlenített algoritmust ismertetünk.

2.2.1. Kiválasztás n^2 processzoron

Legyen $i = n$, azaz a legnagyobb kulcsot keressük. Ez a feladat a következő N´ - algoritmussal n^2 CRCW processzoron $O(1)$ lépéssel elvégezhető.

N´ - ´ (K, y) *párhuzamos eljárás*

Számítási modell: CRCW PRAM

Bemenet: k_1, k_2, ..., k_n (n különböző kulcs)

Kimenet: y (a maximális kulcs értéke)

01 **if** $n = 1$ **then**
02 $y := x_1$
03 P_{ij} **in parallel for** $1 \le i, j \le n$
kiszámítja a $x_{ij} = k_i < k_j$ értéket.
04 Az n^2 processzort n csoportba $(G_1, \ldots, G_n)$ osztjuk úgy, hogy a G_i csoportba a $P_{i,1}, \ldots, P_{i,n}$ processzorok kerüljenek. Mindegyik csoport logikai **or** műveletet végez az $x_{i1}, \ldots, x_{in}$ logikai változókkal.
05 Ha a G_i csoport számítási eredménye a 4. lépésben **false**, akkor a csoport P_{i1} processzora megadja $(y = k_i)$-t kimenetként.

Legyenek a bemenő adatok $k_1, \ldots, k_n$. Az összehasonlításokat párhuzamosan végezzük a P_{ij} $(1 \leq i, j \leq n)$ processzorokon úgy, hogy P_{ij} az $x_{ij} = (k_i < k_j)$ logikai értéket számítja ki. Feltehetjük, hogy a kulcsok különbözőek. Ha mégse, k_i helyett a (k_i, i) párt alkalmazva különbözővé tehetők: ehhez minden kulcshoz egy $(\lg n)$-bites számot kell hozzáadni. Ekkor egyetlen olyan kulcs van, amelyikre minden összehasonlítás eredménye **false**. Ez a kulcs egy logikai **or** művelettel azonosítható.

2.10. tétel. (*Kiválasztás $O(1)$ lépéssel.*) *n kulcs közül a maximális $O(1)$ lépéssel meghatározható n^2* CRCW *közös* PRAM *processzoron.*

Bizonyítás. Az első és a harmadik szakasz egységnyi ideig tart. A második szakasz $O(1)$ lépéssel elvégezhető. ∎

Ennek az algoritmusnak a relatív sebessége $\Theta(n)$. Az elvégzett munka $\Theta(n^2)$. Ezért a hatékonyság $\frac{\Theta(n)}{\Theta(n^2)} = \Theta(\frac{1}{n})$. Tehát az algoritmus nem munkaoptimális.

2.2.2. Kiválasztás p processzoron

Most megmutatjuk, hogy a maximális elem p közös CRCW processzoron $O(\lg\lg p)$ lépéssel meghatározható. A technika az *oszd meg és uralkodj*. Az egyszerűség kedvéért feltesszük, hogy p négyzetszám.

Legyenek a bemenő adatok $X = k_1, k_2, \ldots, k_p$. Legyen az algoritmusunk lépésszáma $T(p)$. A bemenő adatokat $\sqrt{p} = a$ csoportra osztjuk úgy, hogy minden csoportban a elem legyen. Minden csoporthoz rendeljünk q processzort – ekkor a csoportok maximális eleme párhuzamosan számítható. Mivel csoportonként q elem és ugyanannyi processzor van, a csoport maximális eleme $T(a)$ lépéssel meghatározható. Legyenek $M_1, M_2, \ldots, M_a$ a csoportok maximális elemei. Ezek maximuma lesz az algoritmus kimenete. Mivel most csak q elemünk van, az összes processzort alkalmazhatjuk.

A következő CRCW algoritmus $O(\lg\lg p)$ lépést tesz.

G ¨ ¨ - (X, y) *párhuzamos rekurzív eljárás*

Számítási modell: CRCW PRAM

Bemenet: $k_1,\ k_2,\ \ldots,\ k_p$ (p különböző kulcs)

Kimenet: y (a maximális kulcs értéke)

if $p = 1$ **then**
    $y := x_1$
Osszuk a bemenetet $\sqrt{p} = a$ részre $(K_1, K_2, \ldots, K_a)$ úgy, hogy
    K_i a $k_{(i-1)a+1}, k_{(i-1)a+2}, \ldots, k_{ia}$ elemeket tartalmazza. Hasonlóképpen csoportosítsuk a processzorokat úgy, hogy a P_i $(1 \le i \le a)$ csoportba a $P_{(i-1)a+1}, P_{(i-1)a+2}, \ldots, P_{ia}$ processzorok tartozzanak A P_i csoport rekurzívan meghatározza a K_i csoport maximális elemét.
Ha a csoportok maximális elemei $M_1, M_2, \ldots, M_a$, ezek
    maximumát határozzuk meg az előző
    N´ - algoritmussal
    és ez lesz az eredmény.

2.11. tétel. (*Kiválasztás $O(\lg\lg p)$ lépéssel.*) *A* G ¨ ¨ - *algoritmus p közös* CRCW PRAM *processzoron $O(\lg\lg p)$ lépéssel meghatározza p kulcs közül a legnagyobbat.*

Bizonyítás. Ennek az algoritmusnak az első lépése $T(\sqrt{p})$, második lépése $O(1)$ ideig tart. Ezért $T(p)$ kielégíti a

$$T(p) = T(\sqrt{p}) + O(1) \tag{2.7}$$

rekurzív egyenletet, melynek megoldása $O(\lg \lg p)$. ■

A G ¨ ¨ - algoritmus összes munkája $\Theta(p \lg \lg p)$, ezért hatékonysága $\frac{\Theta(p)}{\Theta(p \lg \lg p)} = \Theta(\frac{1}{\lg \lg p})$, így ez az algoritmus sem munkahatékony.

2.2.3. Kiválasztás egész számok között

Legyen a feladat ismét n kulcs maximumának meghatározása. Ha a kulcsok egyetlen bitből állnak, akkor a maximum keresése visszavezethető a logikai VAGY műveletre és ezért $O(1)$ lépéssel meghatározható. Ebből adódik a kérdés: mekkora intervallumban lehetnek a kulcsok ahhoz, hogy p processzoron konstans lépéssel meg tudjuk határozni a maximális elemet?

Legyen c adott konstans, a kulcsok pedig legyenek a $[0, n^c]$ intervallumban. Ekkor a kulcsok legfeljebb $c \lg n$ bites bináris számok. Az egyszerűség kedvéért feltesszük, hogy pontosan ennyi bitesek (a számok elejére szükség esetén nullákat írunk).

A következő CRCW algoritmus $O(1)$ lépést tesz.

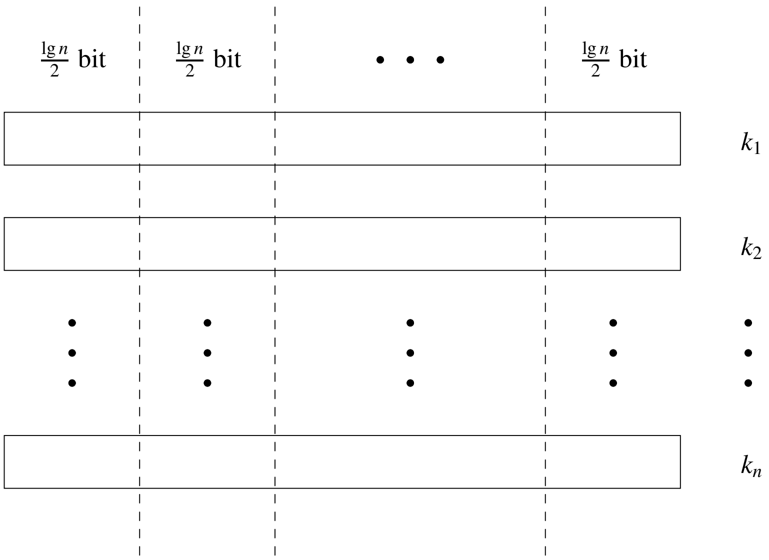
Az alapötlet az, hogy a számok $b_1, b_2, \ldots, b_{2c}$ bitjeit $\frac{\lg n}{2}$ hosszúságú *részekre* (♣) bontjuk. Az i-edik rész a $b_{(i-1)+1}, b_{(i-1)+2}, \ldots, b_{(i-1)+(\lg n)/2}$ biteket tartalmazza, a részek száma $2c$.

Ezt a helyzetet mutatja a 2.5. ábra: először az ábra első oszlopában lévő bitek alapján keressük a maximális kulcsot.

E ´ - ´ (X, y) *párhuzamos eljárás*

Számítási modell: CRCW PRAM

Bemenet: $X = k_1,\ k_2,\ \ldots,\ k_n$ (n különböző kulcs – egész számok)



2.5. ábra. Maximális egész szám kiválasztása

Kimenet: y (a maximális kulcs értéke)

for $i := 1$ **to** $2c$
    Határozzuk meg a megmaradt kulcsok maximumát i-edik
    részük alapján. Legyen a maximum M.
    Hagyjuk el azokat a kulcsokat, melyek i-edik része
    kisebb, mint M.
Az y kimenet legyen a megmaradt kulcsok egyike.

2.12. tétel. (*Kiválasztás egész számok közül.*) *Ha a kulcsok a* $[0, n^c]$ *intervallumból vett egész számok, akkor* p *kulcs közül a maximális* $O(1)$ *lépéssel meghatározható* p CRCW PRAM *processzoron tetszőleges* c *konstans esetében.*

Bizonyítás. Tegyük fel, hogy a kulcsok maximumát a $\frac{\lg n}{2}$ legfontosabb bit alapján határozzuk meg.

Legyen az első részben a maximum M. Ekkor azok a kulcsok, melyek legfontosabb bitjei nem M-et adnak, biztosan nem maximálisak. Ezt az alaplépést megismételjük $2c$-szer, azaz minden $\frac{\lg p}{2}$ bitre pontosan egyszer. Legalább egy kulcs megmarad az utolsó lépés után is – az lesz az eredmény. Az utolsó rész lehet rövidebb, mint $\frac{\lg p}{2}$ bit.

Ha egy kulcs legfeljebb $\frac{\lg n}{2}$-bites, akkor az értéke legfeljebb $\sqrt{n} - 1$. Ezért az E ´ - ´ első lépésében a $[0, \sqrt{n} - 1]$ intervallumba eső egész kulcsok maximumát kell meghatározni. Rendeljünk minden kulcshoz egy processzort és használjunk $\sqrt{n}$ közös memóriarekeszt ($M_1, M_2, \ldots, M_{\sqrt{n}-1}$), melyek tartalma kezdetben $-\infty$. Egy párhuzamos lépésben a P_i processzor k_i-t ír az M_{k_i} memóriarekeszbe. Ezután az n kulcs maximuma a $\sqrt{n}$ memóriarekesz tartalmából n processzorral a 2.9. tétel alapján konstans idő alatt meghatározható. ■

2.2.4. Az általános kiválasztási feladat megoldása $n^2/\lg n$ processzoron

Tegyük fel, hogy az $X = k_1, k_2, \ldots, k_n$ sorozat különböző kulcsokat tartalmaz és az i-edik legkisebb kulcsot akarjuk kiválasztani. Legyen most az x_i kulcs *rangja* (♣) eggyel nagyobb, mint a nála kisebb kulcsok száma (ez a definíció eggyel nagyobb értéket ad, mint a korábban használt).

Ezt a rangot a 2-5. gyakorlat szerint $\frac{n}{\log n}$ CREW PRAM processzoron bármely kulcsra $O(\lg n)$ lépésben meg tudjuk határozni.

Ha $\frac{n^2}{\lg n}$ processzorunk van, akkor azokat $C_1, C_2, \ldots, C_n$ csoportokba oszthatjuk úgy, hogy minden csoportban $\frac{n}{\lg n}$ processzor legyen. A C_j $(1 \le j \le n)$ csoport $O(\lg n)$ lépésben meghatározza a k_j kulcs rangját X-ben. Annak a csoportnak egyik processzora, amelyik az i rangot határozta meg, adja a kimenetet. Az így kapott algoritmus neve legyen Á - ´

2.13. tétel. (*Általános kiválasztás.*) *Az* Á - ´ *algoritmus* $\frac{n^2}{\lg n}$ *processzoron* n *különböző kulcs közül* $\Theta(\lg n)$ *lépésben meghatározza az* i*-edik legkisebbet.*

Nem nehéz belátni, hogy az Á - ´ algoritmus munkája $\Theta(n^2)$, tehát ez az algoritmus sem munkahatékony.

2.2.5. Munkahatékony véletlenített algoritmus ($\star$)

Ebben a pontban $\frac{n}{\log n}$ közös CRCW processzort alkalmazunk arra, hogy $\overline{O}(\lg n)$ lépésben megoldjuk az i-edik legkisebb elem kiválasztását.

A V´ - ´ algoritmus a bemenő kulcsok $X = k_1, k_2, \ldots, k_n$ sorozatából kiválaszt egy $n^{1-\epsilon}$ méretű S mintát és S két elemét elválasztó elemként. Például $\epsilon = 0.6$ megfelelő érték.

Legyen e_1 és e_2 a két elválasztó elem. Az elválasztó elemek olyanok lesznek, hogy a kiválasztandó elem nagy valószínűséggel a két elválasztó elem közé fog esni. Továbbá X-nek az elválasztó elemek közé kevés eleme esik: $\overline{O}(n^{(1+\epsilon)/2}\sqrt{n})$.

Ha már kiválasztottuk a két elválasztó elemet, X-et az $X_1 = \{x \in X | x < e_1\}$, $X_2 = \{x \in X | e_1 \leq x \leq e_2\}$ és $X_3 = \{x \in X | x > e_2\}$ diszjunkt részekre bontjuk. A felbontás során meghatározzuk az egyes részek elemszámát. Ha $|X_1| < i \leq |X_1| + |X_2|$, akkor a kiválasztandó elem X_2 eleme. Ebben az esetben tovább megyünk, míg ellenkező esetben újra kezdjük.

A mintavételezési és kiküszöbölési műveleteket addig ismételjük, amíg el nem érjük, hogy a megmaradó kulcsok száma legfeljebb $n^{0.4}$ legyen. Ezután a megmaradó kulcsok közül az Á - ´ algoritmus segítségével végezzük el a kiválasztást.

Az algoritmus pszeudokódja a következő. A pszeudokódban a N munkaváltozó az élő kulcsok számát adja meg. Kezdetben legyen $N = n$ és minden kulcs legyen *élő* (♣). Minden processzorra $\lg n$ kulcs jut. A 3. és 6. lépésekben a koncentráció azt jelenti, hogy a megfelelő kulcsokat összegyűjtjük és a közös memória egymást követő rekeszeiben helyezzük el őket.

V´ - ´ (X, i, y, n) *párhuzamos eljárás*

Számítási modell: közös CRCW PRAM

Bemenet: $X = k_1,\ k_2,\ \ldots,\ k_p$ (p különböző kulcs)

Kimenet: y (az i-edik legkisebb kulcs értéke)

01 N := n
02 $\epsilon := 0.6$
03 **while** $N > n^0$
04 Minden élő kulcs $\frac{1}{N^\epsilon}$ valószínűséggel kerül az M mintába.
05 P_j **parallel for** $1 \leq j \leq \frac{n}{\lg n}$
P_j meghatározza a minta q méretét és azt minden processzorhoz eljuttatja. Ha nem teljesül $0.5N^{1-\epsilon} \leq q \leq 1.5N^{1-\epsilon}$, akkor folytatja a 01-es lépésnél.
06 Koncentráljuk és rendezzük a mintában lévő kulcsokat.
07 Legyen e_1 az M minta $\lfloor \frac{iq}{N} \rfloor - d\sqrt{q \lg N}$ és e_2 a minta $\lfloor \frac{iq}{N} \rfloor + d\sqrt{q \lg N}$ rangú eleme, ahol d egy $\sqrt{3\alpha}$-nál nagyobb állandó. Szórjuk e_1 és e_2 értékét minden processzorhoz.
08 Számoljuk meg a $[e_1, e_2]$ intervallumba eső élő kulcsok I, valamint az e_1-nél kisebb kulcsok K számát. Minden processzorhoz juttassuk el ezt a két számot. Ha $i \notin (I, I + K)$ vagy $I \neq (N^{(1+\epsilon)/2}\sqrt{N}$, akkor folytassuk az 1-es lépéssel – egyébként hagyjuk el az e_1-nél kisebb és az e_2-nél

nagyobb kulcsokat és legyen $i := i - K$ és $N := I$.
Koncentráljuk és rendezzük a mintában lévő kulcsokat.
Határozzuk meg y értékét.

Nevezzük *fokozatnak* (♣) a **while** ciklus egyszeri lefutását. A minták száma minden fokozatban N és $N^{-\epsilon}$ paraméterekkel rendelkező binomiális eloszlású. Ezért a mintákban lévő kulcsok számának várható értéke $N^{-\epsilon}$. A Csernov-egyenlőtlenség segítségével belátható, hogy

$$|M| = \overline{O}(N^{1-\epsilon}) \tag{2.8}$$

Legyen M egy m elemű minta, amelyet egy n elemű X halmazból állítottunk elő. Legyen $kiválaszt(j, M)$ az M minta j-edik legkisebb eleme és $r_j = rang(kiválaszt(j, M), X)$. Bizonyítás nélkül említjük a következő lemmát.

2.14. lemma. *Minden α számra*

$$P\left(|r_j - j\frac{n}{m}| > \sqrt{3\alpha}\frac{n}{\sqrt{m}\sqrt{\lg n}}\right) < n^{-\alpha}. \tag{2.9}$$

Ennek a lemmának a segítségével bizonyítható a következő állítás.

2.15. tétel. *A* V´ - ´ *algoritmus $\frac{n}{\lg n}$ processzoron $\overline{O}(\lg n)$ lépésben kiválasztja n különböző kulcs közül az i-edik legkisebbet.*

Ebből a tételből és a kiválasztás lineáris lépésigényéből következik, hogy a V´ - ´ algoritmus nagy valószínűséggel munkahatékony.

2.3. Összefésülés

Adott 2 csökkenőleg (vagy növekvőleg) rendezett sorozat, melyek együtt p elemet tartalmaznak. A feladat ennek a sorozatnak egy csökkenő (vagy növekvő) sorozattá való rendezése.

Ez a feladat egy soros processzoron $O(p)$ lépéssel megoldható. Mivel legrosszabb esetben minden elemet meg kell vizsgálni és a helyére kell tenni, ezért a feladat megoldásának lépésszámigénye $\Omega(p)$.

2.3.1. Logaritmikus idejű algoritmus

Legyen $X_1 = k_1, k_2, \ldots, k_m$ és $X_2 = k_{m+1}, k_{m+2}, \ldots, k_{2m}$ a két bemenő sorozat. Az egyszerűség kedvéért legyen m 2 hatványa és a kulcsok különbözzenek.

Az összefésüléshez elég az összes kulcs rangjának kiszámítása. Ha a rangokat ismerjük, akkor $p = 2m$ processzoron egy lépésben beírhatjuk az i rangú kulcsot az i-edik memóriarekeszbe.

2.16. tétel. *A* L -¨ ´ ¨ *algoritmus két m hosszúságú kulcssorozatot $O(\lg m)$ lépéssel összefésül* $2m$ CREW PRAM *processzoron.*

Bizonyítás. A k kulcs rangja legyen r_k^1 (r_k^2) X_1-ben (X_2-ben). Ha $k = k_j \in X_1$, akkor legyen $r_k^1 = j$. Ha egy külön π processzort rendelünk k-hoz, akkor az bináris kiválasztással $O(\lg m)$ lépéssel meghatározza azon X_2-beli elemek q számát, amelyek kisebbek, mint k. Ha q ismert, akkor π kiszámíthatja k $(X_1 \cup X_2)$-beli rangját: ez $j + q$ lesz. Ha k X_2-höz tartozik, hasonlóképpen járhatunk el.

Összegezve: ha elemenként egy, azaz összesen $2m$ processzorunk van, akkor két m hosszúságú rendezett sorozat $O(\lg m)$ lépéssel összefésülhető. Az ezt megoldó algoritmus neve L -¨ ´ ¨ . ■

Ez az algoritmus nem munkahatékony.

2.3.2. Páratlan-páros összefésülő algoritmus

Ez az algoritmus a klasszikus *oszd meg és uralkodj* elvet alkalmazza.

Legyen $X_1 = k_1, k_2, \ldots, k_m$ és $X_2 = k_{m+1}, k_{m+2}, \ldots, k_{2m}$ a két bemenő sorozat. Az egyszerűség kedvéért legyen m kettő hatványa és a kulcsok legyenek különbözőek.

P´ - ´ -¨ ´ ¨ (X_1, X_2, Y) *párhuzamos rekurzív eljárás*

Számítási modell: EREW PRAM

Bemenet: X_1 és X_2 (két rendezett sorozat)

Kimenet: Y (az összefésült sorozat)

if $m = 1$ **then** Fésüljük össze a sorozatokat egyetlen összehasonlítással.

if $m > 1$ **then** Bontsuk fel X_1-et és X_2-t páros és páratlan részre, azaz legyen $X_1^{ptn} = k_1, k_3, \ldots, k_{m-1}$ és $X_1^{prs} = k_2, k_4, \ldots, k_m$. Hasonlóképpen bontsuk fel X_2-t is X_2^{ptn} és X_2^{prs} részekre.

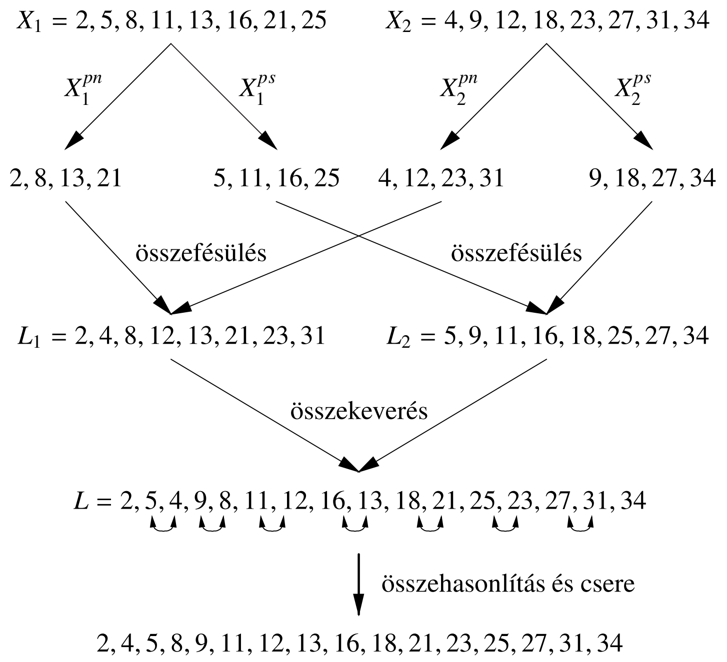
Rekurzívan fésüljük össze X_1^{ptn}-t és X_2^{ptn}-t m processzoron. Legyen az eredmény $L_1 = l_1, l_2, \ldots, l_m$. Ugyanakkor fésüljük össze X_1^{prs}-t és X_2^{prs}-t másik m processzoron: az eredmény legyen $L_2 = l_{m+1}, l_{m+2}, \ldots, l_{2m}$.

Fésüljük össze az L_1, L_2 sorozatokat, azaz legyen $L = l_1, l_{m+1}, l_2, l_{m+2}, \ldots, l_m, l_{2m}$.

Hasonlítsuk össze az $(l_{m+i},\ l_{i+1})$ $(i = 1, 2, \ldots, m-1)$ párokat és szükség esetén cseréljük fel őket. Az eredmény lesz a kimenő sorozat.

2.17. példa. Kétszer nyolc szám összefésülése

Legyen $X_1 = 2, 5, 8, 11, 13, 16, 21, 25$ és $X_2 = 4, 9, 12, 18, 23, 27, 31, 34$. A 16 szám rendezését mutatja a következő 2.6. ábra. ♠



2.6. ábra. 16 szám rendezése a P´ - ´ -¨ ´ ¨ algoritmussal

2.18. tétel. (*Összefésülés $O(\lg m)$ lépéssel.*) *A* P´ - ´ - ¨ ´ ¨ *algoritmus két m hosszúságú kulcssorozatot $O(\lg m)$ lépéssel összefésül $2m$* EREW PRAM *processzoron.*

Bizonyítás. Legyen az algoritmus lépésszáma $M(n)$. Az 1. lépés $O(1)$ ideig tart. A 2. lépés $\frac{m}{2}$ ideig tart. Innen az

$$M(m) = M\left(\frac{m}{2}\right) + O(1) \tag{2.10}$$

rekurzív egyenlőséget kapjuk, melynek megoldása $M(m) = O(\lg m)$. ■

Az összefésülő algoritmusok helyessége a *nulla-egy elv* (♣) segítségével bizonyítható.

Egy összehasonlítás alapú rendező algoritmus *egyszerű* (♣), ha az összehasonlítandó elemek sorozata előre meg van határozva (ekkor a következő összehasonlítás elemei nem függnek a mostani eredménytől).

Formálisan ez azt jelenti, hogy adott az összehasonlítandó elempárok indexeinek $(i_1, j_1),\ (i_2, j_2), \ldots,\ (i_m, j_m)$ sorozata.

2.19. tétel. (*Nulla-egy elv.*) *Ha egy egyszerű összehasonlításos rendező algoritmus helyesen rendez egy n hosszúságú nulla-egy sorozatot, akkor tetszőleges kulcsokból álló n hosszúságú sorozatot is helyesen rendez.*

Bizonyítás. Legyen $\mathcal{A}$ egy egyszerű összehasonlításos (növekvőleg) rendező algoritmus és legyen S egy olyan kulcssorozat, melyet az adott algoritmus rosszul rendez. Ekkor a rosszul rendezett S sorozatban van olyan kulcs, amely az i-edik $(1 \leq i \leq n-1)$ helyen van annak ellenére, hogy S-ben legalább i nála kisebb elem van.

Legyen k S legelső (legkisebb indexű) ilyen kulcsa. A bemenő sorozatban írjunk a k-nál kisebb elemek helyére nullát, a többi elem helyére egyest. Ezt a módosított 0-1 sorozatot $\mathcal{A}$ helyesen rendezi, ezért a k helyére írt egyest a rendezett sorozatban legalább i darab nulla megelőzi.

Most kihasználjuk, hogy $\mathcal{A}$ egyszerű. A bemenő sorozatban színezzük pirosra a k-nál kisebb (nulla) elemeket, és kékre a többit (egyeseket). Indukcióval megmutatjuk, hogy az eredeti és a 0-1 sorozatnak megfelelő színes sorozatok minden összehasonlítás után azonosak. A színek szerint háromféle összehasonlítás van: kék, piros vagy különböző színű elemek összehasonlítása. Ha azonos színű elemeket hasonlítunk össze, akkor a színek sorozata egyik esetben sem változik. Ha viszont különböző színű elemeket hasonlítunk össze, akkor mindkét esetben a piros elem kerül a kisebb, és a kék elem a nagyobb indexű helyre. Eszerint k-t legalább i nála kisebb elem megelőzi a rendezett sorozatban. Az ellentmondás az állítás helyességét mutatja. ■

2.20. példa. Egy nem összehasonlításos rendező algoritmus

Legyen $k_1, k_2, \ldots, k_n$ egy bitsorozat. Rendezhetjük úgy, hogy megszámoljuk a nullák z számát, majd leírunk előbb z nullát, majd $n - z$ egyest. Erre az elv nem alkalmazható, mert ez nem összehasonlításos rendezés. ♠

Az összefésülés viszont rendezés, és a páros-páratlan összefésülés egyszerű.

2.21. tétel. *A* P´ - ´ -¨ ´ ¨ *algoritmus helyesen rendez tetszőleges számokból álló sorozatokat.*

Bizonyítás. Legyenek X_1 és X_2 rendezett 0-1 sorozatok, melyek közös hossza m. Legyen q_1 (q_2) az X_1 (X_2) elején álló nullák száma. Az X_1^{ptn}-ban lévő nullák száma $\lceil \frac{q_1}{2} \rceil$, és az X_1^{prs}-ban lévő nullák száma $\lfloor \frac{q_1}{2} \rfloor$. Így az L_1-beli nullák száma $z_1 = \lceil \frac{q_1}{2} \rceil + \lceil \frac{q_2}{2} \rceil$ és az L_2-beli nullák száma $z_2 = \lfloor \frac{q_1}{2} \rfloor + \lfloor \frac{q_2}{2} \rfloor$.

z_1 és z_2 különbsége legfeljebb 2. Ez a különbség pontosan akkor kettő, ha q_1 és q_2 is páratlan. Egyébként a különbség legfeljebb 1. Tegyük fel, hogy $|z_1 - z_2| = 2$ (a többi eset hasonló). Most L_1-ben kettővel több nulla van. Amikor ezeket a harmadik lépésben összekeverjük, akkor L elején nullák vannak, azután 1,0, majd egyesek. A rendezetlen (*piszkos*) rész csak az 1,0. Amikor a harmadik lépésben az utolsó összehasonlítás és csere megtörténik, az egész sorozat rendezetté válik. ■

2.3.3. Munkahatékony algoritmus

Most $\lceil \frac{2m}{\lg m} \rceil$ processzoron $O(\lg m)$ lépéssel végezzük az összefésülést. Ez a H ´ - ¨ ´ ¨ algoritmus az eredeti problémát $O(\frac{m}{\lg m})$ részre osztja úgy, hogy mindegyikben $O(\lg m)$ hosszúságú rendezett sorozatokat kell összefésülni. Ezek a részproblémák soros algoritmussal $O(\lg m)$ lépéssel megoldhatók.

Legyen $X_1 = x_1, x_2, \ldots, x_m$ és $X_2 = x_{m+1}, x_{m+2}, \ldots, x_{m+m}$ a két bemenő sorozat. Osszuk X_1-et $\lceil \frac{m}{\lg m} \rceil$ részre: ekkor mindegyikben legfeljebb $\lceil \lg m \rceil$ kulcs lesz. A részek legyenek $A_1, A_2, \ldots, A_M$, ahol $M = \frac{m}{\lg m}$. Az A_1-beli legnagyobb kulcs legyen l_i ($i = 1, 2, \ldots, M$). Rendeljünk egy-egy processzort ezekhez az l_i elemekhez. Ezek a processzorok bináris kiválasztással meghatározzák l_i X_2-beli (rendezés szerinti) helyét. Ezek a helyek felbontják X_2-t M részre (ezek között üres részek is lehetnek – lásd a következő 2.7. ábrát). Jelöljük ezeket a részeket B_1, B_2, $\ldots$, B_M-mel. B_i-t az A_1-nek X_2-ben megfelelő részhalmaznak nevezzük.

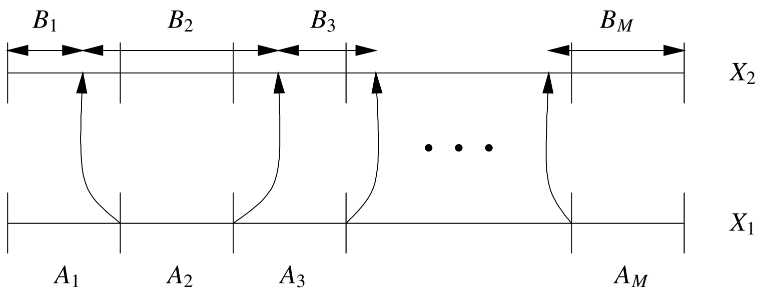
Ekkor X_1 és X_2 összefésülését megkaphatjuk úgy, hogy rendre összefésüljük A_1-et B_1-gyel, A_2-t B_2-vel és így tovább, majd ezeket a sorozatokat egyesítjük.

2.22. tétel. *A* H ´ -¨ ´ ¨ *két m hosszúságú rendezett kulcssorozatot $O(\lg m)$ lépésben összefésül $\lceil \frac{2m}{\lg m} \rceil$* CREW PRAM *processzoron.*

Bizonyítás. Az előző algoritmust alkalmazzuk.

Az A_i részek hossza $\lg m$, a B_i részek hossza azonban nagy is lehet. Ezért még egyszer alkalmazzuk a felbontást. Legyen A_i, B_i tetszőleges pár. Ha $|B_i| = O(\lg m)$, akkor A_i és B_i egy processzoron $O(\lg m)$ lépésben alatt összefésülhető. Ha viszont $|B_i| = \omega(\lg m)$, akkor osszuk B_i-t $\frac{|B_i|}{\lg m}$ részre — ekkor minden rész legfeljebb $\lg m$ egymást követő kulcsot tartalmaz. Mindegyik részhez rendeljünk egy processzort, és az keresse meg az ennek a sorozatnak megfelelő részhalmazt A_i-ben: ehhez $O(\lg \lg m)$ lépés elegendő. Így A_i és B_i összefésülése $\frac{|B_i|}{\lg m}$ részproblémára redukálható, ahol minden részprobléma két $O(\lg m)$ hosszúságú sorozat összefésülése.

A felhasznált processzorok száma $\sum_{i=1}^{M} \left\lceil \frac{|B_i|}{\lg m} \right\rceil$, ami legfeljebb $m/\lg m + M$, és ez legfeljebb $2M$. ■



2.7. ábra. Munkahatékony összefésülő algoritmus

2.3.4. Egy $O(\lg\lg m)$ idejű algoritmus

Ha az előző algoritmust kiegészítjük az *oszd meg és uralkodj* elvvel, akkor még gyorsabb algoritmust kapunk.

G -¨ ´ ¨ (X_1, X_2, Y) *párhuzamos eljárás*

Számítási modell: CREW PRAM

Bemenet: X_1 és X_2

Kimenet: Y

01 Bontsuk fel X_1-et $\sqrt{m} = b$ részre: ekkor minden részben b elem lesz. Legyen A_i-ben a legnagyobb kulcs l_i $(i = 1, 2, \ldots, b)$. Minden l_i-hez rendeljünk b processzort. Ezek a processzorok b-áris keresést végeznek X_2-ben, hogy megtalálják l_i X_2-beli helyét. Ezzel X_2 b részre való felbontását kapjuk: legyenek ezek a részek $B_1, B_2, \ldots, B_b$. A B_i részhalmaz az A_i-nek X_2-ben megfelelő részhalmaz.

02 Most X_1 és X_2 összefésüléséhez elegendő A_i és B_i $(i = 1, 2, \ldots,$ $b)$összefésülése. Az A_i-k mérete adott, viszont a B_i-k nagyon nagyok (és nagyon kicsik) is lehetnek. Ezért újra felbontunk.

Legyen A_i és B_i tetszőleges pár. Ha $|B_i| = O(b)$, akkor a két sorozat $O(1)$ lépésben összefésülhető m^{ϵ}-áris kereséssel (ahol ϵ tetszőleges pozitív szám). Ha viszont $|B_i| = \omega(b)$, akkor B_i-t $\left\lceil \frac{|B_i|}{b} \right\rceil$ részre osztjuk, ahol B_i-nek minden részben legfeljebb b egymást követő eleme van. Rendeljünk minden részhez b processzort, hogy megtalálják az ehhez a halmazhoz tartozó részhalmazt A_i-ben: ehhez $O(1)$ lépés elég. Így A_i és B_i összefésülésének problémája $\left\lceil \frac{|B_i|}{b} \right\rceil$ részproblémára redukálható, ahol minden részprobléma két $O(b)$ hosszúságú sorozat összefésülése.

A felhasznált processzorok száma $\sum_{i=1}^{b} \left\lceil b\frac{|B_i|}{b} \right\rceil$, ami legfeljebb $2m$.

2.23. tétel. (*Összefésülés $O(\lg\lg m)$ lépésben.*) *Két m hosszúságú rendezett kulcssorozat $O(\lg\lg m)$ lépésben összefésülhető* $2m$ CREW PRAM *processzoron.*

Bizonyítás. Legyen X_1 és X_2 a két adott sorozat. Legyenek a kulcsok különbözőek és legyen $\sqrt(m) = b$. Az algoritmus a problémát $N \leq 2b$ részproblémára redukálja, melyek mindegyike két $O(b)$ hosszúságú rendezett sorozat összefésülése. A redukció m processzoron $O(1)$ lépésben elvégezhető. Ha az algoritmus lépésszáma $2m$ processzoron $T(m)$, akkor $T(m)$ kielégíti a $T(m) = T(O(b)) + O(1)$ rekurzív egyenletet, melynek megoldása $O(\lg\lg m)$. ■

Ez az algoritmus nem munkahatékony, bár $\frac{\Theta(m)}{\Theta(\lg\lg m)} = \Theta(\frac{m}{\lg\lg m})$ relatív sebessége nagyon közel van m-hez. Hatékonysága csak $\Theta(\frac{1}{\lg\lg m})$.

2.4. Rendezés

Adott $n \geq 2$ kulcs. A feladat ezek csökkenő vagy növekvő sorrendbe való rendezése.

Ismert, hogy ha a megengedett művelet a szokásos összehasonlítás, akkor minden $\mathcal{A}$ soros algoritmusnak $N(n, \mathcal{A}) = \Omega(n \lg n)$ lépésre van szüksége, másrészt vannak $O(n \lg n)$ lépésszámú összehasonlítás alapú algoritmusok, amelyek tehát aszimptotikusan optimálisak.

Más műveletek vagy a rendezendő kulcsok speciális tulajdonságai esetében a rendezés $O(n)$ lépéssel is megoldható. Ha legrosszabb esetben minden kulcsot meg kell vizsgálni, akkor természetesen a lépésszám $N(n) = \Omega(n)$.

Tehát mind az összehasonlítás alapú, mind pedig a speciális esetben ismert aszimptotikusan optimális soros algoritmus.

Vizsgáljuk meg a következő kérdéseket. Hány rendező algoritmus van? Ezek közül hány egyszerű, hány optimális (aszimptotikusan, szigorúan)?

Ezekre a kérdésekre nem könnyű válaszolni – például először pontosan definiálnunk kell, mi is az a rendező algoritmus.

Szűkítsük a kérdést: hány összehasonlításra van szükség n elem rendezéséhez? jelöljük ezt a számot $c(n)$-nel. Ismert, hogy

$$\left\lceil \sum_{i=1}^{n} \lg i \right\rceil \le c(n) \le \sum_{i=1}^{n} \lceil \lg i \rceil \tag{2.11}$$

és hogy

$$c(n) \le n \lg n - (n-1). \tag{2.12}$$

Az alsó becslés döntési fákkal vagy információelméleti eszközökkel igazolható (lásd az alsó korlátokról szóló 1.7. alfejezetet), a felső becslések pedig a bináris beszúró, illetve az összefésüléses rendező jellemző adatai.

4 elemre az alsó és a felső becslések egyaránt ötöt adnak. 5 elemre $5! = 120$ miatt az alsó becslés 7, viszont az előbbi algoritmusoknak 8 összehasonlításra van szüksége.

n^2 processzoron a kulcsok rangja $O(\lg n)$ lépéssel meghatározható. Ha a rangokat ismerjük, akkor a rendezés egy párhuzamos írással megoldható.

Tehát igaz a következő tétel.

2.24. tétel. (*Rendezés $O(\lg n)$ lépésben.*) *n kulcs n^2* CREW PRAM *processzoron* *rendezhető* $O(\lg n)$ *lépéssel.*

Mivel a kulcsok meghatározása $\Omega(\lg n)$ ideig tart, ez a módszer $\Theta(n^2 \lg n)$ munkát igényel, azaz nem munkahatékony.

2.4.1. Páratlan-páros algoritmus

Ez az algoritmus a klasszikus *oszd meg és uralkodj* elvet alkalmazza. Az egyszerűség kedvéért legyen n kettő hatványa és a kulcsok legyenek különbözők.

A következő EREW PRAM algoritmus $O(\lg^2 n)$ lépést tesz.

P´ - ´ - (X, Y) *párhuzamos rekurzív eljárás*

Számítási modell: EREW PRAM

Bemenet: X (a rendezendő kulcsok)

Kimenet: Y (a rendezett kulcsok)

if $n = 1$ **then**
    $Y := X$
if $n = 1$ **then**
    Osszuk az X bemenetet két részre: ezek legyenek
    $X'_1 = k_1, k_2, \ldots, k_{n/2}$ és
    $X'_2 = k_{n/2+1}, k_{n/2+2}, \ldots, k_n$.
    Rendezze $\frac{n}{2}$ processzor rekurzívan X'_1-t. Az
    eredmény legyen X_1. Ugyanakkor rendezze $\frac{n}{2}$
    processzor rekurzívan X'_2-t.
    Az eredmény legyen X_2.
    Fésüljük össze X_1-et és X_2-t $2m$ processzoron
    a P´ - ´ -¨ ´ ¨ algoritmussal.

Ez az algoritmus hasonlít a soros összefésüléses algoritmusra. Ott azonban a sorozatok legkisebb elemeit hasonlítjuk össze és a kisebb elem az összehasonlítás eredménye.

2.25. tétel. (*Rendezés* $O(\lg^2 n)$ *lépéssel.*) *n kulcs n* EREW PRAM *processzoron rendezhető* $O(\lg^2 n)$ *lépéssel.*

Bizonyítás. Legyen $T(n)$ az algoritmus lépésszáma. Az 1. lépés $O(1)$ ideig tart, a 2. lépés $T(\frac{n}{2})$ ideig, a 3. lépés pedig $O(\lg n)$ ideig. Ezért $T(n)$ kielégíti a

$$T(n) = O(1) + T(\frac{n}{2}) + O(\lg n) \tag{2.13}$$

rekurzív egyenlőséget, melynek megoldása $T(n) = O(\lg^2 n)$. ■

2.26. példa. Rendezés 16 processzorral

Rendezzük 16 processzoron a következő számokat: 25, 21, 8, 5, 2, 13, 11, 16, 23, 31, 9, 4, 18, 12, 27, 34. Az első lépésben a páros és páratlan részeket kapjuk meg, majd a másodikban az első 8 processzor a páratlan részből kapja az $X_1 = 2, 5, 8, 11, 13, 16, 21, 25$-öt, a második 8 processzor pedig az $X_2 = 4, 9, 12, 18, 23, 27, 31, 34$-et. A harmadik lépésben kapjuk az eredményt. ♠

Ez az algoritmus $\Theta(n \lg^2 n)$ munkát végez. Hatékonysága $\Theta(\frac{1}{\lg n})$, gyorsítása $\Theta\left(\frac{n}{\lg n}\right)$.

2.4.2. Egy véletlenített algoritmus ($\star$)

Az $O(\lg^2 n)$ lépésszám véletlenített algoritmussal is elérhető.

2.27. tétel. (*Rendezés* $\overline{O}(\lg^2 n)$ *lépéssel.*) *n kulcs n* CREW PRAM *processzoron rendezhető* $\overline{O}(\lg^2 n)$ *lépéssel.*

Bizonyítás. Ismert, hogy n elem közül $n/\lg n$ processzoron a kiválasztás $\overline{O}(\lg n)$ lépéssel megoldható. Legyen n processzorunk. Ekkor n adott kulcs k mediánja $\overline{O}(\lg n)$ lépéssel megtalálható. Osszuk X-et 2 részre: X_1' tartalmazza a k-nál nem nagyobb kulcsokat, X_2' pedig a többi kulcsot.

Az X_1' és X_2' részeket rekurzívan rendezzük $n/2$ processzoron, majd az eredményeket láncoljuk össze.

Ha a rendezés ideje $T(n)$, akkor

$$T(n) = T\left(\frac{n}{2}\right) + \overline{O}(\lg n), \tag{2.14}$$

melynek megoldása $T(n) = \overline{O}(\lg^2 n)$. ∎

2.4.3. Preparata algoritmusa

Több processzorral a lépésszám csökkenthető: Preparata algoritmusa $n \lg n$ CREW PRAM processzoron $\lg n$ párhuzamos lépést végez.

P (X, Y) *párhuzamos rekurzív eljárás*

Számítási modell: CREW PRAM

Bemenet: X (rendezendő kulcsok)

Kimenet: Y (rendezett kulcsok)

01 Ha n kis szám, akkor rendezzük a bemenetet tetszőleges
 rendező algoritmussal és fejezzük be.

02 Osszuk az adott n kulcsot $\lg n$ részre úgy, hogy mindegyikben
 $\frac{n}{\lg n}$ kulcs legyen. Rendezzük a részeket külön,
 rekurzívan, mindegyik részhez n processzort rendelve. A
 rendezett sorozatok legyenek $S_1, S_2, \ldots, S_{\lg n}$.

03 Fésüljük össze S_i-t és S_j-t $(1 \le i, j \le \lg n)$ párhuzamosan.

04 Rendeljünk $\lg n$ processzort a kulcsok
eredeti sorozatra vonatkozó rangjának meghatározásához.
Végül a kulcsok a rangok sorrendjében adják a kimenetet.

Ez az algoritmus *oszd meg és uralkodj* elvű. A kulcssorozatot $\lg n$ részre osztjuk, majd a részeket páronként összefésülve minden kulcsnak minden részre nézve meghatározzuk a rangját. Ezután a kulcsok tényleges rangja az előbbi rangok összege.

Ha a harmadik lépésben minden (i, j) párhoz $\frac{n}{\lg n}$ processzort rendelünk, akkor az összefésülés $O(\lg \lg n)$ lépéssel elvégezhető.

A negyedik lépésben a rang számítása párhuzamosan végezhető a harmadik lépésben kapott $\lg n$ rang összeadásával: ez a prefixet számító algoritmussal $O(\lg \lg n)$ lépéssel megoldható.

2.28. tétel. (*Rendezés $O(\lg n)$ lépéssel.*) *A* P *algoritmus n elemet $n \lg n$* CREW PRAM *processzoron $O(\lg n)$ lépéssel rendez.*

Bizonyítás. Legyen a Preparata-algoritmus lépésszáma $T(n)$. Az első lépés lépésigénye $T(\frac{n}{\lg n})$, a második és harmadik lépésé együtt $O(\lg \lg n)$. Ezért

$$T(n) = T\left(\frac{n}{\lg n}\right) + O(\lg \lg n), \tag{2.15}$$

melynek megoldása (helyettesítéses módszerrel) $T(n) = O(\lg n)$. ■

A lassulásra vonatkozó tétel segítségével kapjuk a következő állítást.

2.29. következmény. (*Rendezés $\frac{n \lg n}{t}$ processzoron.*) *Tetszőleges $t \geq 1$ egész számra n tetszőleges kulcs rendezhető $O(t \lg n)$ lépésben $\frac{n \lg n}{t}$* CREW PRAM *processzoron.*

A Preparata-algoritmus munkája ugyanannyi, mint a páros-páratlan rendező algoritmusé, viszont a gyorsítása jobb: $\Theta(n)$. Mindkét algoritmus hatékonysága $\Theta(\frac{1}{\lg n})$.

2.4.4. Reischuk véletlenített algoritmusa ($\star$)

Reischuk algoritmusa n processzoron $\overline{O}(\lg n)$ párhuzamos lépést tesz, azaz nagy valószínűséggel munkahatékony.

Alapja Preparata algoritmusa és a következő tétel.

2.30. tétel. (*Korlátos egészek rendezése.*) *Ha n kulcs mindegyike a* $[0, n(\lg n)^c]$ *intervallumból vett egész szám (ahol c tetszőleges konstans), akkor ezek a kulcsok* $\frac{n}{\lg n}$ CREW PRAM *processzoron rendezhetők* $\overline{O}(\lg n)$ *lépésben.*

Az algoritmus a következő.

R (X, Y) *párhuzamos eljárás*

Számítási modell: CREW PRAM

Bemenet: X (rendezendő kulcsok)

Kimenet: Y (rendezett kulcsok)

01 Legyen $X = k_1, k_2, \ldots, k_n$ a rendezendő kulcssorozat.
   $N = \frac{n}{\lg^4 n}$ processzor mindegyike véletlenül kiválaszt egy kulcsot.
02 Rendezzük az N kiválasztott kulcsot Preparata algoritmusával.
   Az eredmény legyen $l_1, l_2, \ldots, l_N$.
03 Legyen $K_1 = \{k \in X | k \leq l_1\}$;
   $K_i = \{k \in X | l_{i-1} < k \leq l_i\}$, $i = 2, 3, \ldots, N$;
   és $K_{N+1} = \{k \in X | k > l_N\}$. Párhuzamos bináris kiválasztással
   bontsuk fel X-et K_i részekre.
04 Rendezzük a K_i részeket a P algoritmussal.
05 Legyen Y a rendezett K_1, rendezett K_2, ..., rendezett K_{N+1}.

2.31. tétel. (*Tetszőleges kulcsok gyors rendezése.*) *A* R *algoritmus n tetszőleges kulcsot n* CREW PRAM *processzoron $\overline{O}(\lg n)$ lépésben rendez.*

Bizonyítás. Reischuk algoritmusa szerint véletlenül kiválasztunk $N = \frac{n}{\lg^4 n}$ kulcsot és ezeket Preparata algoritmusával rendezzük. Ez a rendezett minta az eredeti sorozatot $N + 1$ – közel azonos hosszúságú – részsorozatra bontja, melyeket Preparata algoritmusával rendezünk: ezek a rendezett részsorozatok a megfelelő sorrendben véve megadják az eredményt.

A R algoritmus második lépése $N \lg N \leq N \lg n$ processzoron $O(\lg N) = O(\lg n)$ idő alatt elvégezhető.

A harmadik lépésben X felbontását bináris kereséssel és az egészeket rendező algoritmussal végezzük: minden kulcshoz tartozik egy processzor, amely az $l_1, l_2, \ldots, l_N$ sorozatban végzett bináris kereséssel megadja, hogy az adott kulcs a K_i részhalmazok melyikéhez tartozik.

Mivel a részek sorszámai a $[0, N + 1]$ intervallumból vett egészek, az előző tétel alapján legfeljebb n processzoron $\overline{O}(\lg n)$ lépéssel rendezhetők. A K_i ($1 \leq i \leq N$) részekben nagy valószínűséggel nem lesz több elem, mint $O(\lg^5 n)$. Ugyanennyi processzoron és lépéssel a $|K_i|$ elemszámokat is meg tudjuk határozni minden részre.

A negyedik lépésben minden K_i rész rendezhető $|K_i| \lg |K_i|$ processzoron $\lg |K_i|$ lépéssel. Ezért a negyedik lépés n processzoron

$$(\max_i \lg |K_i|)^2) \tag{2.16}$$

lépéssel elvégezhető. Ha $\max_i |K_i| = O(\lg^5 n)$, akkor a negyedik lépés $O((\lg \lg n)^2)$ lépéssel elvégezhető. ■

2.5. Gráfalgoritmusok

Legyen M egy $n{\times}n$ méretű mátrix, amelynek az elemei nemnegatív egész számok. Az $\widetilde{M}$ mátrixot M *minmátrixának* (♣) nevezzük és a következőképpen definiáljuk:

$$\widetilde{M}\,(i,i) = 0 \;\; (1 \le i \le n) \tag{2.17}$$

és

$$\widetilde{M}\,(i,j) = \min\{M_{i_0 i_1} + M_{i_1 i_2} + \ldots + M_{i_{k-1} i_k}\} \;\; (i \ne j), \tag{2.18}$$

ahol a minimumot az olyan – az $\{1, 2, \ldots, n\}$ halmaz elemeiből képezett – $i_0, i_1, \ldots, i_k$ permutációkra nézve képezzük, amelyekre $i_0 = i$ és $i_k = j$.

Legyen G egy irányítatlan gráf, amelynek n csúcsa van. Ennek reflexív tranzitív lezártját A^*-gal jelöljük és úgy definiáljuk, hogy ha $i = j$ vagy V_i és V_j G-nek ugyanahhoz az összefüggő komponenséhez tartoznak, akkor $A^*(i,j) = 1$, egyébként $A^*(i,j) = 0$.

2.32. példa. Irányított gráf és minmátrixa

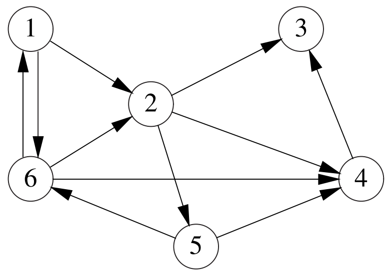
Legyen $G(V,E)$ egy irányított gráf, melynek csúcsait az $1, 2, \ldots, n$ számokkal címkézzük. Legyen $M[i,j] = 0$, ha $i = j$ vagy a gráf tartalmaz i-ből j-be vezető élet. Egyébként $M[i,j]$ legyen 1. Ekkor $\widetilde{M}\,(i,j)$ pontosan akkor nulla, amikor van az i csúcsból a j csúcsba vezető út.

A 2.8. ábra egy 6-csúcsú irányított gráfot, valamint a hozzá tartozó M mátrixot és annak minmátrixát ábrázolja.

♠

2.5.1. Minmátrix

Több – gráfokkal kapcsolatos – feladat megoldását megkönnyíti a minmátrixok felhasználása.



M

0	0	1	1	1	0
1	0	0	0	0	1
1	1	0	1	1	1
1	1	0	0	1	1
1	1	1	0	0	0
0	0	1	0	1	0

$\widetilde{M}$

0	0	0	0	0	0
0	0	0	0	0	0
1	1	0	1	1	1
1	1	0	0	1	1
0	0	0	0	0	0
0	0	0	0	0	0

2.8. ábra. Egy gráf és annak M-je és $\widetilde{M}$

A M´ algoritmus meghatározza adott M mátrix minmátrixát.

M´ $(M, \widetilde{M})$ *párhuzamos eljárás*

Számítási modell: CREW PRAM

Bemenet: M (egy $n \times n$ méretű mátrix)

Kimenet: $\widetilde{M}$ (az M mátrix minmátrixa)

P_{ij} **in parallel for** $1 \leq i, j \leq n$
    $m[i, j] := M[i, j]$
Rendezzük az N kiválasztott kulcsot Preparata algoritmusával.
    Az eredmény legyen $l_1, l_2, \ldots, l_N$.

```
for i = 1 to n
    P_ijk in parallel for 1 ≤ i, j, k ≤ n
        q[i, j, k] := m[i, j] + m[j, k]
    P_ij in parallel for 1 ≤ i, j ≤ n
        m[i, j] := min{q[i, 1, j], q[i, 2, j], . . . , q[i, n, j]}
P_i in parallel for 1 ≤ i ≤ n
    M̃ (i, j) := 0
P_ij in parallel for 1 ≤ i, j ≤ n
    M̃ (i, j) := m(i, j)
```

Ezzel kapcsolatos a következő állítás.

2.33. tétel. (*Minmátrix számítása.*) *Ha* ϵ *tetszőleges pozitív szám, akkor a* M - ´ *algoritmus* $n^{3+\epsilon}$ CRCW PRAM *processzoron* $O(\lg n)$ *lépésben meghatározza egy* $n \times n$ *méretű mátrix minmátrixát.*

2.5.2. Tranzitív lezárt

A 2.33. tétel segítségével belátható a következő állítás.

2.34. tétel. (*Tranzitív lezárt számítása.*) *Ha* ϵ *tetszőleges pozitív szám, akkor* $n^{3+\epsilon}$ CRCW PRAM *processzoron* $O(\lg n)$ *lépésben meghatározható egy* n*-csúcsú irányított gráf tranzitív lezártja.*

Bizonyítás. Ha M-et a 2.5. példa szerint definiáljuk, akkor egy G gráf tranzitív lezártja a minmátrix segítségével könnyen meghatározható. ■

2.5.3. Összefüggő komponensek

Ugyancsak a 2.33. tétel segítségével bizonyíthatjuk a következő állítást.

2.35. tétel. (*Összefüggő komponensek számítása.*) *Ha* ϵ *tetszőleges pozitív szám, akkor* $n^{3+\epsilon}$ CRCW PRAM *processzoron* $O(\lg n)$ *lépésben meghatározhatók egy* n*-csúcsú gráf összefüggő komponensei.*

Bizonyítás. Legyen $M[i, j] = 0$, ha $i = j$ vagy i és j össze vannak kötve egy éllel. Egyébként $M[i, j]$ legyen 1. Az i és j csúcsok pontosan akkor vannak ugyanabban az összefüggő komponensben, ha $\widetilde{M}[i, j] = 0$. ■

2.5.4. Minimális feszítőfa

A soros Kruskal-algoritmus párhuzamosításával belátható a következő állítás.

2.36. tétel. (*Minimális feszítőfa.*) *Ha* ϵ *tetszőleges pozitív szám, akkor* $n^{5+\epsilon}$ CRCW PRAM *processzoron* $O(\lg n)$ *lépésben meghatározható egy* n*-csúcsú élsúlyozott gráf minimális feszítőfája.*

2.6. Gyakorlatok

2-1. A globális memória M_1 rekeszében van bizonyos adat. Másoljuk át ezt az adatot az $M_2, M_3, \ldots, M_n$ rekeszekbe. Mutassuk meg, hogyan lehet ezt megvalósítani $O(\lg n)$ lépéssel n EREW PRAM processzor felhasználásával.

2-2. Adjunk meg egy olyan algoritmust, amely $\frac{n}{\lg n}$ PRAM processzor felhasználásával $O(\lg n)$ lépéssel megoldja az előző gyakorlatot.

2-3. Legyen $f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$. Adjunk $O(1)$ idejű CREW PRAM algoritmust a polinom értékének adott x helyen való kiszámítására. Mennyi processzort igényel a javasolt algoritmus?

2-4. Adjunk meg egy $O(\lg \lg n)$ idejű algoritmust, amely $\frac{n}{\lg \lg n}$ közös CRCW PRAM processzoron $O(\lg \lg n)$ lépésben megadja n tetszőleges szám maximumát.

2-5. Legyen A egy n kulcsot tartalmazó tömb. Mutassuk meg, hogy $\frac{n}{\lg n}$ CREW PRAM processzoron $O(\lg n)$ lépésben meghatározható tetszőleges $k \in A$ kulcs rangja.

2-6. Tervezzünk egy $O(1)$ lépésszámú algoritmust, amely n közös CRCW PRAM processzoron eldönti, hogy adott $A[1:n]$ tömb elemei között előfordul-e az 5, és ha igen, megadja a legnagyobb olyan i indexet, amelyre $A[i] = 5$.

2-7. Tervezzünk algoritmust, amely n^2 CREW PRAM processzoron $O(1)$ lépésben összefésül két n hosszúságú rendezett sorozatot.

2-8. Határozzuk meg a fejezetben tárgyalt algoritmusok relatív sebességét, összes lépésszámát és hatékonyságát.

2.7. Feladatok

2-1. Közös elem

Tervezzünk algoritmust annak eldöntésére, hogy adott $A[1:n]$ és $B[1:n]$ tömböknek van-e közös eleme:

- ha n^2 CRCW PRAM processzorunk van, akkor $O(1)$ lépésszámút;
- ha n CRCW PRAM processzorunk van, akkor $\overline{O}(1)$ lépésszámút.

2-2. Minimális feszítőfa

Párhuzamosítsuk a minimális feszítőfák meghatározására szolgáló Kruskal-algoritmust és Prim-algoritmust. Tervezzünk algoritmust arra a speciális esetre, amikor az élek súlya csak 0 vagy egy lehet.

2-3. Összes csúcspár távolsága

Párhuzamosítsuk a gráfok összes csúcspárjának távolságát meghatározó Bellman-Ford algoritmust.

2-4. Körmentesség

Tervezzünk párhuzamos algoritmust annak eldöntésére, hogy adott irányítatlan gráf tartalmaz-e kört. Elemezzük a különböző nagyságrendű processzorszám esetében elérhető $W(N, p, \mathcal{P})$ lépésszámokat.

3. Rácsok

Ebben a fejezetben rácsokat (láncot, négyzetet és kockát) alkalmazunk számítási modellként.

3.1. Számítási modellek

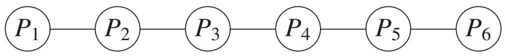
A k-dimenziós ($k \geq 1$) rács egy olyan $m_1 \times m_2 \times \cdots \times m_k$ ($m_1, m_2, \ldots, m_k \geq 2$) méretű háló, amelynek minden egyes metszéspontjában van egy processzor. Az élek a kommunikációs vonalak, melyek kétirányúak. A rács egy processzorát megcímkézzük egy $(i_1, i_2, \ldots, i_k)$ k-assal, – erre a processzorra a $P_{i_1,\ldots,i_k}$ jelöléssel hivatkozunk.

Minden processzor egy RAM, amely rendelkezik saját (helyi) memóriával. A $P_{i_1,\ldots,i_k}$ processzor saját memóriája az $M[i_1, \ldots, i_k, 1]$, $M[i_1, \ldots, i_k, 2]$, $\ldots$, $M[i_1, \ldots, i_k, m]$ rekeszekből áll.

Minden processzor végre tud hajtani egyetlen lépésben olyan alapvető műveleteket, mint az összeadás, kivonás, szorzás, összehasonlítás, saját memória elérése és így tovább. A processzorok működése szinkron módon történik, azaz minden processzor egy globális óra ütemére egyszerre hajtja végre az aktuális feladatát.

A legegyszerűbb rács a $k = 1$ értékhez tartozó lánc alakú rács (röviden *lánc*) (♣).

Egy 6-processzoros lánc látható a 3.1. ábrán.

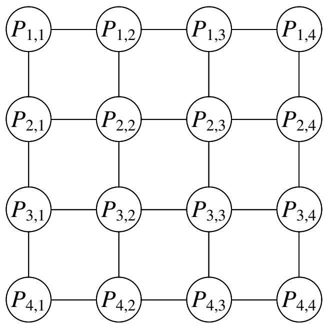


3.1. ábra. 6-processzoros lánc

Egy lánc processzorai P_1, P_2, $\ldots$, P_p. Ezek a következőképpen vannak összekötve. P_1 és P_p kivételével mindegyik processzor össze van kötve a nála eggyel nagyobb (jobb szomszéd), illetve eggyel kisebb (bal szomszéd) indexűvel, míg a két szélső processzornak csak egy szomszédja van, P_2, illetve P_{p-1}. Az összeköttetés kétirányú.

Ha $k = 2$, akkor téglalap alakú rácsot kapunk. Ha most $m_1 = m_2 = \sqrt{p} = a$, akkor $a \times a$ méretű *négyzetet* (♣) kapunk.

Egy 4×4 méretű négyzet látható a 3.2. ábrán.



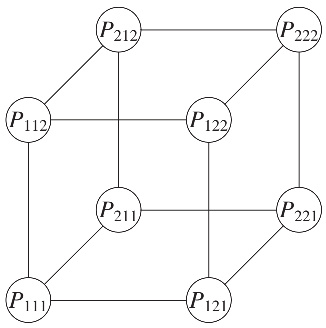
3.2. ábra. 4×4 méretű négyzet

Egy $a \times a$ méretű négyzet tartalmaz részgráfként számos a processzoros láncot. A rács algoritmus egyes lépései gyakran tekinthetők láncokon végzett műveleteknek. A processzorok közötti kommunikáció bármely rögzített összekötöttségű gépben kommunikációs láncok segítségével történik. Ha két olyan processzor akar kommunikálni egymással, amik egy éllel össze vannak kötve, akkor azt egyetlen lépésben elvégezhetik. Ha nincs közöttük él, akkor az őket összekötő utak valamelyikén történhet meg a kommunikáció, tehát a szükséges lépésszám (legalábbis rövid üzenetek esetén) függ az út hosszától. Feltesszük, hogy egy processzor egyetlen lépésben képes végrehajtani egy számítást és/vagy kommunikálni akár mind a négy szomszédjával.

Egy rácsban azok a processzorok, amelyeknek első (második) koordinátája megegyezik, egy sort (oszlopot) alkotnak. Például egy $a \times a$ méretű négyzetben az i-edik sor a $P_{i,1},\ P_{i,2},\ \ldots,\ P_{i,a}$ processzorokból áll. Mindegyik sor vagy oszlop egy a processzoros lánc. Egy rács algoritmus gyakran áll olyan lépésekből, melyeket csak bizonyos sorokban vagy oszlopokban lévő processzorok végeznek el.

Ha $k = 3$, akkor tégla alakú rácsot kapunk. Az $a_1 = a_2 = a_3 = \sqrt[3]{p}$ speciális esetben *kockáról* (♣) beszélünk és a kocka méretére az $n \times n \times n$ jelölést alkalmazzuk.

A 3.3. ábra egy $2 \times 2 \times 2$ méretű kockát ábrázol.



3.3. ábra. $2 \times 2 \times 2$ méretű kocka

3.2. Csomagirányítás

A processzorok közötti kommunikáció egyetlen lépése egy rögzített szerkezetű hálózatban a következő – *csomagirányítási problémának* (♣) nevezett – feladatként fogható fel. A hálózatban minden processzornak van egy adatcsomagja, amit egy másik processzornak akar elküldeni. A feladat a csomagok eljuttatása a céljukhoz a lehető leggyorsabban úgy, hogy egy lépésben egy kommunikációs csatornán egy irányban egyszerre csak egy csomag utazhat. Az utóbbi feltételre a csatorna sávszélességének korlátozottsága miatt van szükség. Könnyen előfordulhat, hogy egy adott lépésben kettő vagy több csomag érkezik egy processzorhoz, és mindegyik ugyanazon a csatornán szeretne továbbhaladni. Ilyen esetben természetesen csak egy csomag utazhat a következő lépésben, a többiek pedig a processzornál egy várakozási sorba kerülnek a későbbi továbbküldés miatt. Egy *elsőbbségi szabály* (♣) alapján döntjük el, hogy melyik csomagot küldjük el ilyen esetekben. Ilyen elsőbbségi szabályok például az FDF (legtávolabbra utazó csomag először), FOF (legtávolabbról jött csomag először), FIFO (előbb érkezett csomag először), RAN (véletlenül választunk).

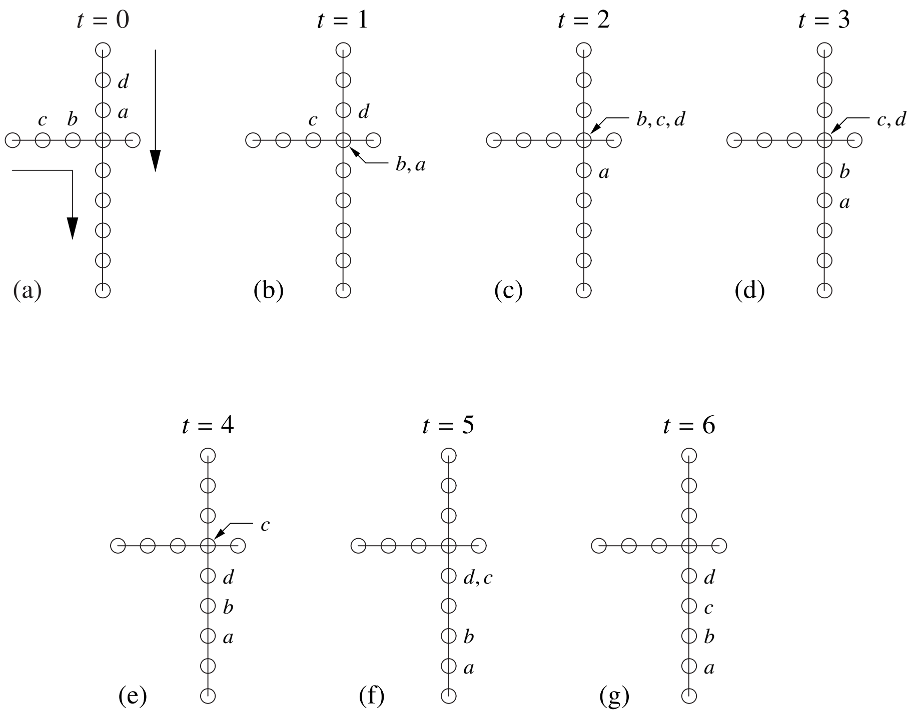
A csomagirányítási probléma egy speciális esete a *parciális permutációs csomagirányítás* (PPR = partial permutation routing) (♣). A PPR-ben minden processzornak legfeljebb egy elküldendő csomagja van, és egy adott processzorhoz legfeljebb egy csomagot kell küldenünk. Vegyük észre, hogy a PPR-t egy EREW PRAM gépen egy párhuzamos írással megoldhatjuk. De egy általános rögzített összekötöttségű hálózatban a probléma gyakran igen bonyolult. Tipikusan a bemenet egy bizonyos sorrendben kerül a processzorokhoz, és a kimenetnek úgyszintén egy előre megadott sorrendben kell megjelennie. Csupán egy ilyen sorrend átrendezés néha több PPR-t igényelhet. Ezért bármely nem triviális rögzített összekötöttségű hálózati algoritmus tervezéséhez szükség van PPR-ekre. Ez az egyik lényeges különbség a hálózati és a PRAM algoritmusok között.

Egy csomagirányítási algoritmus legfontosabb jellemzői a *lépésszáma* (♣), amíg az utolsó csomag is eléri a célját, és a *várakozási sor hossza* (♣), ami az egy processzornál maximálisan várakozó csomagok száma. A sor lehetséges hosszát alulról korlátozza az egy processzortól induló, valamint az egy processzorhoz érkező csomagok maximális száma. Feltételezzük, hogy a csomag nemcsak a küldendő információt tartalmazza, hanem a küldő és a célprocesszor azonosítóját is. Egy csomagirányítási algoritmus bemenő adatai a csomagok indulási helye és célja, valamint a használni kívánt *elsőbbségi szabály*. Egy csomag utazásának lépésszáma az indulás és a cél között megtett út és a sorokban eltöltött várakozás hosszától függ.

3.1. példa. 4 csomag irányítása

Tekintsük az a, b, c, d csomagokat a 3.4. ábra (a) részén. A rendeltetési helyüket az ábra (g) része mutatja. Használjuk a FIFO elsőbbségi szabályt, ahol holtverseny esetén önkényesen döntsünk valakinek a javára. A csomagok a lehető legrövidebb úton közlekedjenek. A $t = 1$ sorszámú lépésben minden csomag eggyel közelebb kerül a céljához. Ennek eredményeként az a és a b ugyanazon a ponton vannak, tehát a $t = 2$ sorszámú lépésben valamelyiket várakoztatni kell. Mivel mindketten egyszerre érkeztek, ezért ez egy holtverseny. Önkényesen döntünk, ezért nyerjen mondjuk a. Ugyanebben a lépésben még c és d is csatlakozik b-hez. A következő lépésben b fog továbbmenni, mivel ő előbb érkezett, és ezért elsőbbsége van a többihez képest. A negyedik lépésben c és d holtversenyben küzd a továbbhaladásért. Legyen d a győztes, aki tovább haladhat. További két lépésben c is eljut a céljába. Ekkorra minden csomag megérkezett.

A c csomagnak négy élen kellett átutaznia, és két alkalommal várt, azaz a lépésszám 6, mivel c utazott a legtovább. Vajon más elsőbbségi szabályt használva csökkenhet a lépésszám? Játsszuk végig az előző példát az FDF szabállyal. Ekkor a lépésszám 5 lesz. ♠



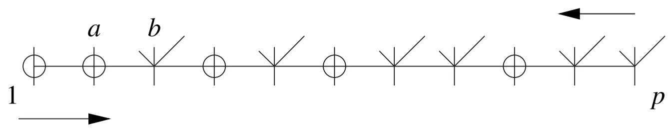
3.4. ábra. Példa csomagirányításra

3.2.1. Csomagirányítás láncon

Egy láncon, mivel az összekötöttség kétirányú, a processzor egyszerre küldhet és fogadhat a szomszédjaitól üzeneteket. Ennek következtében két ellentétes irányú csomaglánc nem zavarja egymást. Ebben a részben megmutatjuk, hogy a PPR láncon megoldható legfeljebb $p-1$ lépésben. Vegyük észre, hogy a legrosszabb esetben legalább $p-1$ lépés kell, hiszen ez a lehető legnagyobb távolság két processzor között. Láncon PPR-en kívül vizsgálunk még néhány általánosabb csomagirányítási feladatot is.

3.2. példa. Független csomagáramok

A 3.5. ábrán a balról jobbra haladó csomagokat körökkel, a jobbról balra haladó csomagokat pedig pipával jelöltük. feltesszük, hogy a balra haladó csomagok célja P_1, a jobbra haladó csomagoké pedig P_p. Például az a és b csomagoknak egyidejűleg van szüksége ugyanarra az élre, amikor az első lépést megteszik (ellenkező irányban). Mivel az élek kétirányúak, nincs verseny, a csomagok használhatják egyszerre az élet. Mivel a P_1 processzortól induló csomagnak $p-1$ élen kell keresztül haladnia, ezért a célba éréshez legalább $p-1$ lépésre van szüksége. ♠



3.5. ábra. A jobbra és balra haladó csomagáramok függetlenek

Tegyük fel, hogy egy p processzorból álló láncban minden processzor legfeljebb egy üzenetet küld, az üzenetek célja tetszőleges lehet. Továbbítsuk a csomagokat a célprocesszorokhoz. Ezt a feladatot *egy kezdőcsomagos feladatnak* (♣) nevezzük.

3.3. lemma. (*Egy kezdőcsomagos feladat.*) *Az egy kezdőcsomagos feladat egy p-processzoros láncon megoldható legfeljebb $p-1$ lépésben.*

Bizonyítás. Minden q üzenetet küldhetünk a kezdő és a végpont közötti legrövidebb úton. Tekintsük csak azokat az üzeneteket, amelyek balról jobbra utaznak, hiszen a jobbról balra utazókat teljesen hasonlóan függetlenül vizsgálhatjuk. Ha q a P_i processzortól indul és P_j felé tart, akkor $j-i$ lépésben éri azt el, hiszen sosem kell várakoznia egy másik üzenetre. A leghosszabb ilyen út 1-től p-ig vezet, ezért

$p - 1$ felső korlát a lépésszámra. Az egy csúcsba küldött csomagok maximális száma jellemzi az algoritmus várakozási sorának hosszát. ■

Adott egy p processzorból álló lánc. A P_i processzor k_i $(0 \leq k_i \leq p)$ üzenetet akar küldeni és

$$\sum_{i=1}^{p} k_i = p. \tag{3.1}$$

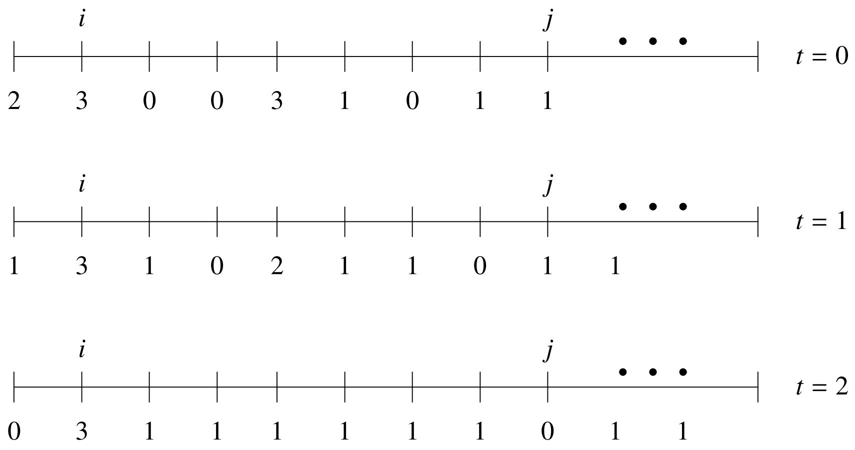
Nincs két olyan üzenet, amelyeket azonos processzorhoz kellene küldeni. Továbbítsuk a csomagokat a célprocesszorokhoz. Ezt a feladatot *egy célcsomagos feladatnak* (♣) nevezzük.

3.4. lemma. (*Egy célcsomagos feladat.*) *Ha az FDF elsőbbségi szabályt használjuk, akkor az egy célcsomagos feladat egy p-processzoros láncon megoldható legfeljebb* $p - 1$ *lépés alatt.*

Bizonyítás. Tegyük fel, hogy q üzenet P_i-ből P_j-be megy. Az általánosság megszorítása nélkül feltehetjük, hogy a csomag balról jobbra halad. A jobbról balra haladó csomagoktól eltekinthetünk hasonló okok miatt, mint az előző lemmánál. Minden csomag a lehető legrövidebb úton halad, ami a mi esetünkben $j - i$ lépést jelent. Ne feledkezzünk meg azonban azokról a csomagokról, amelyek j-nél nagyobb indexű processzorokhoz utaznak, mivel ezek (és csak ezek) megvárakoztathatják q-t. Ezen csomagok száma legfeljebb $p - j$. Jelölje $k_1, k_2, \ldots, k_{j-1}$ a kezdeti állapotban rendre a P_1, P_2, $\ldots$, P_{j-1} processzortól induló ilyen csomagok számát. Jelölje m minden lépés után azt az indexet, melyre $k_{m-1} > 1$, és ha $m \leq s \leq j$, akkor $k_s \leq 1$. Nevezzük a $k_m, k_{m+1}, \ldots, k_{j-1}$ sorozatot *szabad sorozatnak* (♣).

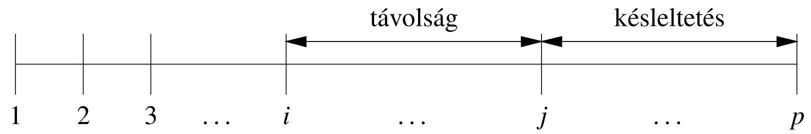
Vegyük észre, hogy az elkövetkezőkben a szabad sorozatban egyik csomag sem fog várakozni. Ezenfelül minden egyes lépésben legalább egy csomag csatlakozik a szabad sorozathoz.

A 3.6. ábra egy szabad sorozatot mutat. Az ábrán a számok a megfelelő processzoroknál (melyeknek csak az indexe szerepel az ábrán) lévő csomagok számát mutatják. Például a nulladik lépésben ($t = 0$) lépésben P_i-nél 3 csomag van és 1, 0, 1, 1 a szabad sorozat. A következő lépésben ($t = 1$) egy, majd az azt követő lépésben ($t = 2$) 4 új csomag csatlakozik a szabad sorozatban lévő csomagokhoz.



3.6. ábra. Szabad sorozat

Így legfeljebb $p - j$ lépés után minden olyan csomag csatlakozott a szabad sorozathoz, amely miatt q várakozásra kényszerülhet. Ezt a helyzetet mutatja a 3.7. ábra: a q csomagnak a legfeljebb $p - j$ lépésnyi várakozáson felül $j - i$ lépést kell tennie, ezért legfeljebb $p - i$ lépés után célba ér.



3.7. ábra. A 3.3. lemma bizonyításának szemléltetése

A jobbról balra haladó csomagok esetében hasonlóan belátható, hogy legfeljebb $i - 1$ lépésben megérkeznek a rendeltetési helyükre. ■

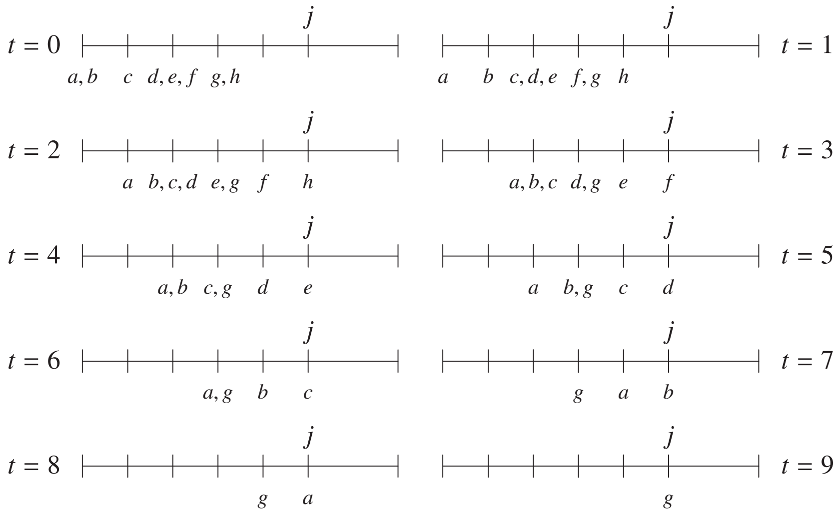
Most definiáljuk az *általános csomagirányítási* (♣) feladatot. Tételezzük fel, hogy egy p processzoros láncon több csomag származhat egy processzortól és több csomagot küldhetünk egy processzorhoz. Továbbá a $P_1, P_2, \ldots, P_j$ ($j = 1, 2, \ldots, p$) processzoroktól összesen induló csomagok száma nem több, mint $j + f(p)$, p valamely rögzített $f(p)$ függvényére. Továbbítsuk a csomagokat a célprocesszorokhoz.

3.5. lemma. (*Általános csomagirányítási feladat.*) *Ha a* FOF *elsőbbségi szabályt használjuk, akkor az általános csomagirányítási probléma megoldható* $p + f(p)$ *lépésben.*

Bizonyítás. Legyen q egy P_i-ből P_j-be utazó üzenet, ahol $i < j$ (a $j > i$ eset hasonlóan kezelhető). A q csomag legfeljebb $i + f(p)$ csomag miatt várakozhat, hiszen csak ennyi csomag érkezhet távolabbról és lehet ezért nagyobb prioritása. Ha q minden egyes ilyen csomag miatt legfeljebb egyszer kényszerül várakozni, akkor az összes várakozási lépésszáma legfeljebb $i + f(p)$. Ha azonban valamely r csomag mondjuk kétszer várakoztatja meg q-t, ez azt jelenti, hogy r várakozásra kényszerült egy másik, nagyobb prioritású csomag miatt, ami sosem fogja q-t megvárakoztatni. Emiatt q várakozása legfeljebb $i + f(p)$ lehet. Mivel $q - nak$ már csak $j - i$ lépést kell megtennie, ezért legfeljebb $j + f(p)$ lépés kell q helyre szállításához. Az összes csomagra ez a lépésszám legfeljebb $p + f(p)$. ■

3.6. példa. A 3.5 lemma bizonyításának szemléltetése

A 3.8. ábra mutatja a 3.5. lemma bizonyításának menetét (a processzoroknak itt is csak az indexe szerepel). A példában 8 csomag van: a, b, c, d, e, f, g, h. Vizsgáljuk a g csomagot. Ezt a csomagot a többi 7 csomag késleltetheti. A g csomag a $t = 9$ sorszámú lépésben éri el célját. Megtett útjának hossza 2, késleltetése 7. Az ábra nem mutatja azokat a csomagokat, amelyek a P_j csúcsban keresztezték egymást. ♠



3.8. ábra. A 3.5. lemma bizonyítását illusztráló példa

3.2.2. Egy mohó algoritmus a PPR megoldására rácson

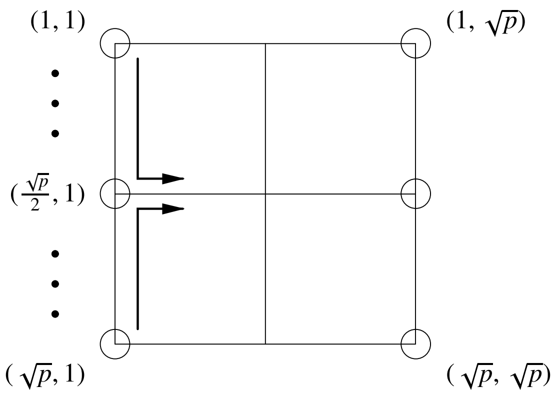
Egy PPR probléma megoldásához egy $\sqrt{p} \times \sqrt{p} = a \times a$ méretű rácson egy üzenetnek az $(1,1)$ csúcsból az (a,a) csúcsba szállításához legalább $N(n, a^2, \mathcal{P}) \geq 2(a-1)$ lépésre van szükség. Ezért $2(a-1)$ egy alsó korlát bármely csomagirányító algoritmus lépésszámának legrosszabb esetét vizsgálva: $W(n, a^2, \mathcal{A}) \geq 2(a-1)$.

Egy egyszerű PPR algoritmus, amely felhasználja a láncoknál ismertetett csomagirányítási algoritmusokat, a következő. Legyen q egy tetszőleges csomag, amelyet a $P_{i,j}$ processzortól a $P_{k,l}$ indexűnek kell elküldeni. Az csomag az első fázisban a j-edik oszlop mentén a k-adik sorig halad a legrövidebb úton. A második fázisban a k-adik sor mentén a legrövidebb úton az l-edik oszlophoz elérve a csomag megérkezett a rendeltetési helyére. Egy csomag azonnal megkezdheti a második fázist az első befejezése után, nem kell semmi másra várnia.

Az első fázis legfeljebb $a-1$ lépésben elvégezhető, mivel a 3.3. lemma alkalmazható. A második fázis a 3.5. lemmából következően nem tart $a - 1$ lépésnél tovább. Így az algoritmus lépésszáma legfeljebb $2(a - 1)$, ami az elméleti alsó korlát, azaz az algoritmus abszolút optimális.

De van egy komoly hátránya ennek az algoritmusnak, mégpedig az, hogy a várakozási sor hossza $\frac{a}{2}$. Nézzünk erre egy példát. Legyen a csomagirányítási probléma olyan, hogy az első oszlop minden csomagját a $\frac{a}{2}$-edik sorba kelljen szállítani. Egy ilyen PPR esetén a $(\frac{a}{2}, 1)$ indexű processzor minden lépésben két csomagot kap. Mivel mindkettő ugyanazt a kommunikációs vonalat szeretné használni, ezért az egyik a várakozási sorba kerül. Ez megismétlődik egészen a $\frac{a}{2}$-edik lépésig, mikor is $\frac{a}{2}$ csomag lesz az $(\frac{a}{2}, 1)$ processzor várakozási sorában.

Ezt a helyzetet mutatja a 3.9. ábra.



3.9. ábra. A mohó algoritmusnak hosszú sorra van szüksége

Az ideális olyan algoritmus lenne, amelynek $O(1)$ méretű várakozási sorra van szüksége (vagy legalább olyan algoritmus, melyre az $f(p)$ függvény lassan nő, mint például a $\lg p$).

3.2.3. Egy kis várakozási sort használó véletlenített algoritmus (⋆)

A kétfázisú mohó algoritmus módosítható a véletlenítés segítségével úgy, hogy csak $\overline{O}(\lg p)$ méretű várakozási sort használjon. Az új H´ - ´ ´ algoritmus három fázisú és a lépésszáma $3a + \overline{o}(a)$. Ebben az algoritmusban a három fázis teljesen elkülönül, azaz egy fázis csak akkor kezdődhet el, ha az előző fázist már minden csomag befejezte. Ez a megszorítás egyszerűbbé teszi az elemzést

Legyen q egy tetszőleges csomag, amelyet $P_{i,j}$-től $P_{k,l}$-hez kell továbbítani.

1. fázis. A q csomag választ egy véletlen $P_{i',j}$ processzort starthelyének oszlopában és a legrövidebb úton eljut oda.

2. fázis. A q csomag az i'. sor mentén továbbítódik a $P_{i',l}$ processzorhoz.

3. fázis. Végül q – az l. oszlop mentén haladva – eléri rendeltetési helyét.

3.7. tétel. *A* H´ - ´ ´ *algoritmus befejeződik* $3\sqrt{p} + \overline{O}(p^{1/4} \lg p)$ *lépés alatt.*

Bizonyítás. A 3.3. lemma alkalmazásából adódik, hogy az első fázis legfeljebb a lépésben befejeződik, mivel egyetlen csomag sem kényszerül várakozásra.

Tegyük fel, hogy egy csomag a második fázist a $P_{i',j}$ processzornál kezdi. Az általánosság megszorítása nélkül feltehetjük, hogy a csomag jobbra mozog. A $P_{i',j}$ processzortól induló csomagok száma ennél a lépésnél egy binomiális eloszlású valószínűségi változó

$$B\left(a, \frac{1}{a}\right) \tag{3.2}$$

paraméterekkel. Azért ennyi, mert a csomag van a j-edik oszlopban és mindegyik $\frac{1}{a}$ valószínűséggel kerül $P_{i',j}$-hez az első fázis végén. Ezenfelül a második fázis kezdetén a $P_{i',1},\ P_{i',2},\ \ldots, P_{i',j}$ processzoroktól induló csomagok száma szintén binomiális eloszlású

$$B\left(ja, \frac{1}{a}\right) \tag{3.3}$$

paraméterekkel. (Felhasználtuk azt a tényt, hogy $B(n_1, x) + B(n_2, x) = B(n_1 + n_2, x)$.) Ennek a változónak a várható értéke j. A Csernov-egyenlőtlenséget felhasználva ez a szám $\geq 1 - p^{-\alpha-1}$ valószínűséggel legfeljebb $j + 3\alpha p^{1/4} \ln p$ minden $\alpha \geq 1$ számra. Így ez az érték $j + (p^{1/4} \lg p)$. A 3.3. lemmát alkalmazva most azt kapjuk, hogy a második fázis $a + \overline{O}(p^{1/4} \lg p)$ lépésben befejeződik.

A harmadik fázis elején bármely oszlopban akármelyik processzortól legfeljebb a csomag indul és legfeljebb egy csomag érkezik. Ezért a 3.5. lemmának megfelelően a harmadik fázis legfeljebb a lépést igényel. ■

3.3. Alapfeladatok

Ebben az alfejezetben négy alapfeladat megoldását mutatjuk be: üzenetszórás, prefixszámítás, adatkoncentráció és ritka rendezés. Egy $a \times a$ méretű rácson mind a négy feladat megoldható $O(a)$ lépésben.

Mivel egy üzenet a négyzetrács egyik sarkából csak $d = 2(a - 1)$ lépésben juthat el az átellenes sarokba, ez a szám a lépésszám alsó korlátja az előbbi feladatokra, és legrosszabb esetben szükség van az átellenes sarokban lévő processzorok közti kommunikációra. A d szám a négyzetrács *átmérője* (♣).

A *k-k irányítási probléma* (♣) a következő. A hálózat bármely processzorától bármely másikhoz legfeljebb k csomagot kell elküldeni.

3.3.1. Üzenetszórás

Az *üzenetszórási feladat* szerint a hálózat megadott processzorától üzenetet kell eljuttatni megadott célprocesszorokhoz (rendszerint az összes többi processzorhoz).

Legyen $\mathcal{L}$ egy p processzoros lánc és legyen M egy üzenet, amelyet a P_1 processzornak kell elküldenie a többi processzorhoz. A feladat megoldható olyan egyszerűen, hogy P_1 elküldi az üzenetet P_2-nek, az tovább küldi P_3-nak és így tovább. Ekkor az üzenet $p-1$ lépésben eljut a legtávolabbi processzorhoz, P_p-hez. Mivel a lánc átmérője $p-1$, ez a lehető legkisebb lépésszám.

Egy $a \times a$ méretű négyzetrácsban az üzenetküldést két fázisban valósíthatjuk meg. Az első fázisban az üzenetet küldő $P_{i,j}$ processzor eljuttatja üzenetét az i-edik sor minden processzorához. Ezután a második fázisban az i-edik sor minden processzora eljuttatja az üzenetet a vele azonos oszlopban lévő processzorokhoz. Ez összesen legfeljebb $2(a-1)$ lépést vesz igénybe.

Formalizáljuk állításunkat.

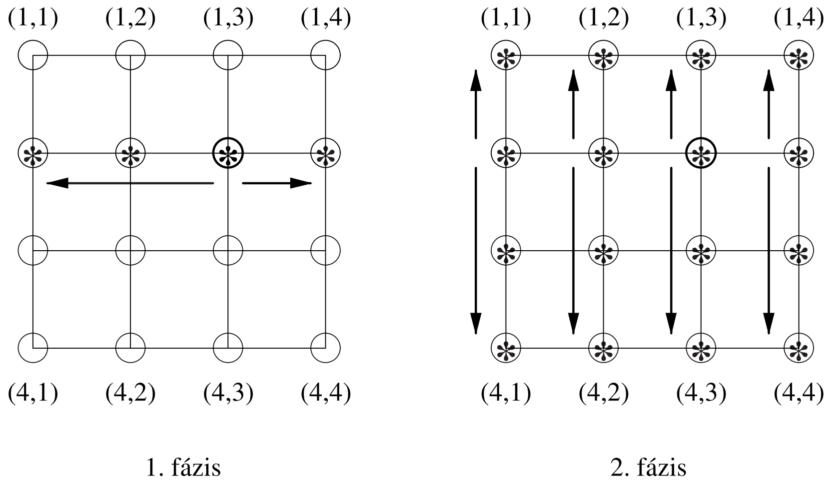
3.8. tétel. *Egy p processzoros láncon az üzenetszórás $p-1 = O(p)$ lépés alatt elvégezhető. Egy $a \times a$ méretű rácson az üzenetszórás $2(a-1) = O(a)$ lépésben elvégezhető.*

3.9. példa. Üzenetszórás rácson

A 3.10. ábra egy 4×4 méretű négyzeten való üzenetszórás két fázisát mutatja. Az üzenetek a $P_{2,3}$ processzortól indulnak és az első fázisban eljutnak a $P_{2,1}$, $P_{2,2}$ és $P_{2,4}$ processzorokhoz. A második fázisban a második sor processzorai mind felfelé, mind lefelé elküldik az M üzenetet. Az üzenetszórás a legrosszabb esetre jellemző 6 lépés helyett már 4 lépés alatt befejeződik. ♠

3.3.2. Prefixszámítás

A második fejezetben több algoritmust ismertettünk, amelyek különböző párhuzamos gépeken megoldják a prefixszámítási feladatot. Most láncon és négyzeten alkalmazható algoritmusok következnek.



3.10. ábra. Üzenetszórás négyzeten

3.3.2.1. Prefixszámítás láncon

Tegyük fel, hogy az $\mathcal{L}$ lánc P_i ($i = 1,\ 2,\ \ldots,\ p$) processzorának saját memóriájában van az x_i elem, és a számítás végén a P_i processzor memóriájában lesz az y_i prefix.

Először nézzünk egy naiv algoritmust.

L´ - (X, Y) *párhuzamos eljárás*

Számítási modell: lánc

Bemenet: $X = x_1,\ x_2,\ \ldots,\ x_p$ (az összeadandó elemek)

Kimenet: $Y = y_1,\ y_2,\ \ldots,\ y_p$ (a prefixek)

P_i **in parallel for** $1 \le i \le p$
    if $i = 1$ **then**
        P_1 az első lépésben elküldi
        x_1-et P_2-nek
    if $i = p$ **then**

P_p a p-edik lépésben kap egy elemet (z_{p-1}-et) P_1-től, kiszámítja és tárolja $z_{p-1} \oplus x_p$-t

if $i \neq 1 \wedge i \neq n$ **then**

P_i az i-edik lépésben kap egy elemet (z_{i-1}-t) P_{i-1}-től, kiszámítja és tárolja $z_{i-1} \oplus x_i$-t

Közvetlenül adódik ennek az algoritmusnak a lépésszáma.

3.10. tétel. *A* L´ - *algoritmus egy láncon* $\Theta(p)$ *lépésben határozza meg* p *elem prefixeit.*

Mivel egy soros processzorral p elem prefixei $O(p)$ lépésben meghatározhatók, ugyanakkor $W(p, p, \text{L´ -}) = p^2$, ezért L´ - nem munkahatékony.

Hasonló algoritmus alkalmazható négyzeten is. Tekintsünk egy $a \times a$ méretű négyzetet. Szükségünk van a processzorok egy *indexelésére* (♣). Ilyenek a *sorfolytonos* (♣), az *oszlopfolytonos* (♣), a *kígyószerű sorfolytonos* (♣) és a *blokkonként kígyószerű sorfolytonos indexelés* (♣).

A blokkonként kígyószerűen sorfolytonos indexelésnél a rácsot megfelelő méretű kisebb blokkokra bontjuk, és a blokkokon belül alkalmazzuk valamelyik indexelést.

A négyzeten való prefixszámítás 3 fázisra osztható, melyekben a számításokat minden fázisban vagy csak soronként, vagy oszloponként végezzük. Az alábbi N´ - algoritmus sorfolytonos indexelést alkalmaz.

N´ - (X, Y) *párhuzamos eljárás*

Számítási modell: négyzet

Bemenet: $X = (x_1,\ x_2,\ \ldots,\ x_p)$ (az összeadandó elemek)

Kimenet: $Y = (y_1,\ y_2,\ \ldots,\ y_p)$ (a prefixek)

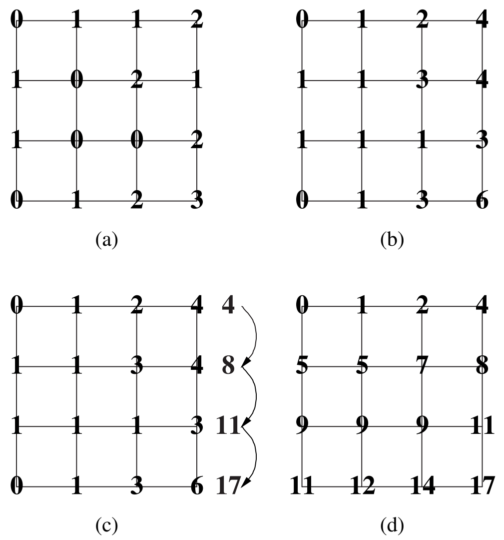
S_i **in parallel for** $1 \leq i \leq a$
  call L´ - $(S_i, X[i], Y[i])$
O_j **in parallel for** **call** L´ - $(O_i, Z[i])$
$P_{j,a}$ **in parallel for** $1 \leq j \leq a-1$
  elküldi a kiszámolt prefixet $P_{j+1,a}$-nak
S_i **in parallel for** $1 \leq i \leq a-1$
  call Ü - ´ ()
P_{ij} **in parallel for** $1 \leq i \leq a-1, 1 \leq j \leq a$
  kiszámolja és tárolja $z_{i,a} \oplus y_{i+1,j}$-t
  kiszámítja és tárolja $z_{i-1} \oplus x_i$-t

A lépésszám a következő.

3.11. tétel. *A* N´ - *algoritmus $a \times a$ méretű négyzeten sorfolytonos indexeléssel $3a + 2 = O(a)$ lépésben elvégzi a prefixszámítást.*

3.12. példa. Prefixszámítás 4×4 méretű négyzeten

A 3.11. ábra (a) részén 16 rendezendő szám látható. Az első fázisban minden sorban kiszámítjuk a prefixeket – az eredményt a (b) rész mutatja. A második fázisban csak a negyedik oszlopban számolunk – az eredmény a (c) részen látható. Végül a harmadik fázisban aktualizáljuk a prefixeket – az eredményt a (d) rész mutatja. ♠



3.11. ábra. Prefixszámítás négyzeten

3.3.3. Adatkoncentráció

Tegyük fel, hogy egy p processzoros hálózatban $d < p$ adat van – processzoronként legfeljebb egy. *Adatkoncentráció* az a feladat, hogy az adatokat egyesével helyezzük el az első d processzornál. Lánc esetében az adatokat a $P_1, P_2, \ldots, P_d$ processzorokhoz kell mozgatni. Rács esetében tetszés szerinti indexelés alkalmazható.

3.13. tétel. *Az adatkoncentráció láncon legfeljebb* $2p$ *lépésben,* $a \times a$ *méretű négyzeten* $6a + \overline{O}(p^{1/4})$ *lépésben megoldható.*

3.3.4. Ritka rendezés

Ha a rendezendő kulcsok száma lényegesen kisebb, mint a rendező hálózat mérete, akkor *ritka leszámláló rendezésről* (♣) beszélünk.

R - (X, Y) *párhuzamos eljárás*

Számítási modell: négyzet

Bemenet: X (a rendezendő kulcsok)

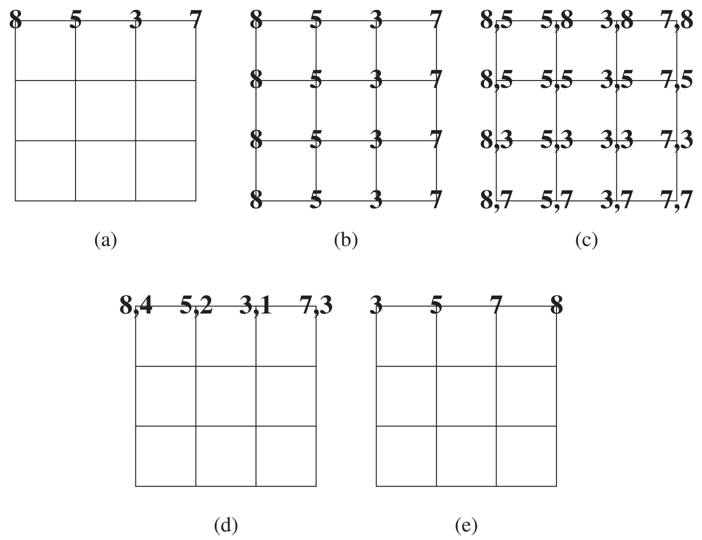
Kimenet: Y (a rendezett kulcsok)

P_i **in parallel for** $1 \leq j \leq a$
  Szórja a k_j kulcsot a j-edik oszlopban
P_i **in parallel for** $1 \leq i \leq a$
  Szórja a k_i kulcsot az i-edik oszlopban
P_i **in parallel for** $1 \leq i \leq a$
  Kiszámítja k_i rangját az i-edik sorban
  prefixszámítást végezve
P_i **in parallel for** $1 \leq j \leq a$
  Elküldi a k_j kulcs rangját $P_{i,j}$-nek
P_r **in parallel for** $1 \leq r \leq a$
  az r rangú kulcs elküldése a $P_{1,r}$ processzorhoz

3.14. tétel. (*Rendezés négyzeten.*) *Ha a rendezendő kulcsok száma legfeljebb a, akkor a* R - *algoritmus rendezés egy négyzeten $O(a)$ lépés alatt befejeződik.*

3.15. példa. 4 kulcs rendezése egy 4×4 méretű négyzeten

A 3.12. ábra mutatja a (8,5,3,7) kulcssorozat rendezését egy 4×4 méretű négyzeten. Az ábra (a) részében a bemenő adatok láthatók (a processzorok első soránál). Az első lépésben az algoritmus az oszlopokban szórja az üzenetet (j-edik oszlopban k_j-t): az eredményt a (b) részen látjuk. A második lépésben soronként szórjuk az üzeneteket (az i-edik sorban k_i-t) – az eredmény a (c) részben látható. A 3. és 4. lépés utáni helyzetet – amikor a kulcsok és rangjaik az első sorban vannak – tükrözi a (d) ábrarész. Az (e) részben a kulcsok már rendezve vannak a processzorok első sorában. ♠



3.12. ábra. Ritka leszámláló rendezés négyzeten

3.4. Kiválasztás

A második fejezetben már foglalkoztunk a speciális és általános kiválasztási feladat megoldására alkalmas PRAM algoritmusokkal.

Négyzeten két változatot vizsgálunk. Az egyikben feltételezzük, hogy a kulcsok és a processzorok száma megegyezik. A második változat szerint a processzorok száma kisebb, mint a kulcsok száma. Ha a számítási modellünk PRAM, akkor az első feladatot megoldó algoritmus és a lassulási tétel segítségével olyan algoritmust kaphatunk, amely hasznosítja az elvégzett munkát. Rácsokra azonban nem ismert ilyen általános, a lassulásra vonatkozó állítás. Ezért a második esetet külön kell kezelni.

3.4.1. Véletlenített algoritmus az $p = n$ esetre ($\star$)

A párhuzamos gépre javasolt H ´ - ´ algoritmus módosítható úgy, hogy négyzeten is munkahatékony legyen – így kapjuk a N´ - - ´ algoritmust.

Erre az algoritmusra érvényes a következő állítás.

3.16. tétel. *A* N´ - - ´ *algoritmus p kulcs esetében egy négyzeten $O(a)$ lépésben megoldja a kiválasztási feladatot.*

Bizonyítás. A *fokozat* a **while** ciklus magjának egyszeri végrehajtása. Korábban megmutattuk, hogy az algoritmus $\overline{O}(1)$ fokozat alatt véget ér.

A H ´ - ´ algoritmus első szakasza rácson $O(1)$ lépést igényel. A 2–5. lépésekből álló szakaszban leírt prefixszámítás $O(a)$ lépés alatt befejeződik. A 2–6. lépésekben végzett adatkoncentrációhoz a 3.11. tétel szerint elég $O(\sqrt{p})$ lépés. A 3. és 6. szakaszban végzett ritka rendezéshez szintén elég $O(\sqrt(p))$ lépés. A 4. és 6. szakaszban végzett kiválasztás konstans lépésben elvégezhető, mivel rendezett sorozatból kell választani. ■

3.4.2. Véletlenített algoritmus a $p < n$ esetre ($\star$)

Tegyük fel, hogy kevesebb processzorunk van, mint kulcsunk. Ha feltesszük, hogy alkalmas $c > 1$ konstanssal teljesül $n = p^c$, akkor a H ´ - ´ algoritmus ennek a feladatnak a megoldására is átalakítható.

Minden processzor $\frac{n}{p}$ kulccsal kezd. A **while** utasítás feltétele $N > D$ lesz (ahol D egy állandó). A második lépésben a processzorok minden hozzájuk tartozó kulcsot $\frac{1}{N^{1-(1/3c)}}$ valószínűséggel veszik a mintához. Ezért ez a lépés $\frac{n}{p}$ időt vesz igénybe. A mintában lévő kulcsok száma $\overline{O}N^{1/3c} = \overline{o}(\sqrt{p})$. A harmadik lépés változatlan és $O(\sqrt{p})$ időt vesz igénybe. Mivel a mintában csak $\overline{O}(N^{1/3c})$ kulcs van, a negyedik lépésben $O(\sqrt{p})$ idő alatt koncentrálhatók és rendezhetők. Az ötödik lépés $O(\sqrt{p})$ ideig tart, a hatodik és hetedik ugyancsak. Így minden fokozat $O(\frac{n}{p} + \sqrt{p})$ ideig tart.

Ennek az algoritmusnak a lépésszámára vonatkozik a következő tétel.

3.17. tétel. *Ha $c > 1$ állandó és $n = p^c$, akkor az n kulcs közül történő kiválasztás egy négyzeten $\overline{O}\left((\frac{n}{p} + \sqrt{p}) \lg\lg p\right)$ lépésben elvégezhető.*

Bizonyítás. A 2.3. lemmából következik, hogy az egyes fokozatokat túlélő kulcsok száma legfeljebb

$$2\sqrt{\alpha}N^{1-(1/6c)}\sqrt{\lg N} = \overline{O}(N^{1-(1/6c)}\sqrt{\lg N}, \tag{3.4}$$

ahol N az élő kulcsok száma az adott fokozat kezdetekor.

Ebből adódik, hogy az algoritmusnak csak $\overline{O}(\lg\lg p)$ fokozata van. ■

3.4.3. Determinisztikus algoritmus a $p < n$ esetre

Az algoritmus alapja, hogy az elemeket például ötös csoportokra bontja, minden csoportnak meghatározza a mediánját, majd kiszámítja ezen mediánok M mediánját. Ezután meghatározzuk M rangját (r_M), majd az $i \leq r_M$ összehasonlítás eredményétől függően elhagyjuk az M-nél nagyobb, illetve nem nagyobb elemeket. Végül a megmaradó kulcsok közül rekurzívan kiválasztjuk a megfelelőt.

Amikor ezt az algoritmust hálózatban hajtjuk végre, akkor célszerű arra törekedni, hogy a kulcsok egyenletesen legyenek a processzorok között elosztva. Ennek érdekében a mediánok M mediánját súlyozással számoljuk: minden csoportot az elemszámának megfelelő súllyal veszünk figyelembe.

Legyen $X = k_1,\ k_2,\ \ldots, k_n$ egy kulcssorozat, és legyen w_i a k_i kulcs súlya, továbbá legyen a súlyok összege W:

$$W = \sum_{i=1}^{n} w_i. \tag{3.5}$$

Ekkor az X sorozat *súlyozott mediánja* (♣) az a $k_j \in X$ kulcs, amelyre

$$\sum_{k_m \in X,\ k_m \leq k_j} w_{k_l} \geq \frac{W}{2} \tag{3.6}$$

és

$$\sum_{k_m \in X,\ k_m \geq k_j} w_{k_l} \geq \frac{W}{2}. \tag{3.7}$$

3.18. példa. Súlyozott médián meghatározása

A súlyozott médián meghatározásának egyik módja, hogy rendezzük a kulcsokat, és az így kapott $X' = k'_1,\ k'_2,\ \ldots,\ k'_n$ kulcssorozathoz tartozó $W' = w'_1,\ w'_2,\ \ldots,\ w'_n$ rendezett súlysorozatnak meghatározzuk a legkisebb olyan prefixét (az összeadás a művelet), amely már legalább $\frac{W}{2}$. Legyen $X = 1, 3, 5, 6, 4, 2$ és a megfelelő súlyok legyenek 1, 2, 3, 4, 5, 6. Ekkor $W = 21$, $W' = 1, 6, 2, 5, 3, 4$ és a prefixek 1, 7, 9, 14, 17, 21. Mivel 14 a legelső megfelelő prefix, ezért X – W-vel súlyozott – mediánja 4 (és ennek súlya 5). ♠

D - ´ - ´ $(k_1, k_2, \ldots, k_p, i)$ *párhuzamos eljárás*

Számítási modell: négyzet

Bemenet: $K = k_1, \ldots, k_p$ (a kulcsok)

Kimenet: i (a kiválasztott kulcs indexe)

```
N := 0
if lg n/p ≤ lg lg p then
                    Minden processzor rendez
```

else
  Minden processzor felosztja a kulcsokat lg p egyenlő
  részre úgy, hogy a kulcsok minden részben
  legfeljebb akkorák, mint a tőle jobbra lévő részben.
while $N > D$
  P_q **in parallel for** $1 \leq q \leq p$
    P_q meghatározza a nála lévő
    kulcsok mediánját. Legyen M_q a
    médián és N_q a P_q-nál
    maradó kulcsok száma ($1 \leq q \leq p$).
  Meghatározzuk $M_1, M_2, \ldots M_p$ súlyozott mediánját úgy,
  hogy M_q súlya N_q. Legyen M a súlyozott médián.
  Meghatározzuk M rangját a maradék kulcsok között
  (legyen ez r_M) és ezt a rangot szétszórjuk.
  if $i \leq r_M$ **then** eltávolítunk minden kulcsot,
    amely M-nél nagyobb.
  Kiszámítjuk és szórjuk az eltávolított kulcsok E számát.
  if $i > r_M$ **then**
  $i := i - E;\ N := N - E$
A kimenet k_i, az i-edik legkisebb kulcs

3.19. példa. Kiválasztás 3×3 méretű rácson

A 3.13. ábra determinisztikus kiválasztást mutat be. ♠

A lépésszám a következő.

3.20. tétel. (*Kiválasztás négyzeten.*) D - ´ - - ´ *n kulcs közül $a \times a$ méretű négyzeten $O\left(\frac{n}{p} \lg\lg p) + a \lg n\right)$ lépésben megoldja a kiválasztást.*

Processzor	(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	(2,3)	(3,1)	(3,2)	(3,3)
	11	18	10	21	12	24	19	15	1
	16	2	17	26	7	4	20	8	13
	3	14	5	27	25	9	23	22	6

A súlyozott médián 14.

11	2	10	-	12	4	-	8	1
3		5		7	9			13
								6

A súlyozott médián 5.

11	-	10	-	12	9	-	8	13
				7				6

A súlyozott médián 8.

A válasz 8.

3.13. ábra. Determinisztikus kiválasztás 3×3 méretű négyzeten

3.5. Összefésülés

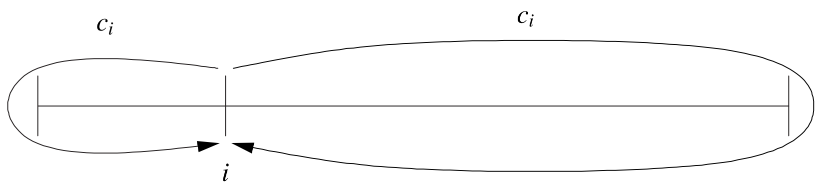
A második fejezetben több algoritmust is elemeztünk, amelyek PRAM segítségével oldották meg az összefésülési feladatot.

3.5.1. Rangon alapuló összefésülés láncon

A R rangsorolási algoritmus átalakítható úgy L´ -¨ ´ ¨ algoritmussá, hogy a lépésszáma láncon lineáris legyen. Legyen $\mathcal{L}$ egy p processzoros lánc. A bemenő sorozatok legyenek $X_1 = k_1, k_2, \ldots, k_m$ és $X_2 = k_{m+1}, k_{m+2}, \ldots, k_{2m}$.

Kezdetben a P_i processzornál két kulcs van: k_i és k_{i+m}.

A 3.14. ábra mutatja, hogyan utaznak a c_i számlálót tartalmazó csomagok, amelyek végül visszatérnek a P_i processzorhoz.



3.14. ábra. Kulcsok rangjának számítása

3.21. tétel. (*Rangsorolásos összefésülés láncon.*) *Két p hosszúságú rendezett sorozat egy p processzoros láncon $O(p)$ lépésben összefésülhető.*

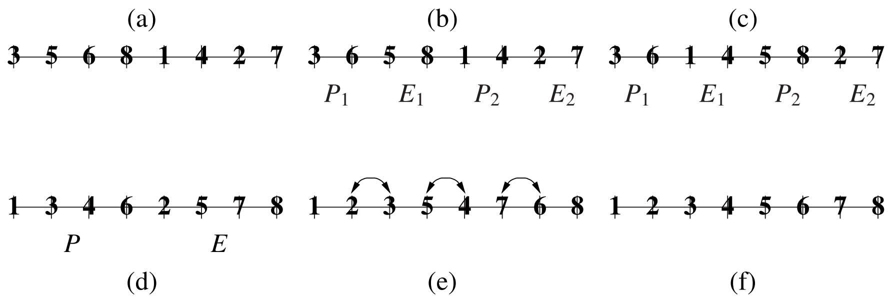
3.5.2. Páratlan-páros összefésülés láncon

3.22. tétel. (*Páros-páratlan összefésülés láncon.*) *Két m hosszúságú sorozat egy $2m$ processzoros láncon $O(m)$ lépésben összefésülhető.*

3.23. példa. Két 4 hosszúságú sorozat összefésülése 8 processzoron

A 3.15. ábra mutatja, hogyan fésüli össze az algoritmus a rendezett (3,5,6,8) és (1,4,2,7) sorozatokat egy 8 processzoros láncon. Az ábra (a) része a bemenő adatokat mutatja. A (b) rész a bemenő sorozatok páros és páratlan indexű elemeket tartalmazó részsorozatokra való felbontását mutatja. A (c) rész az P_2 és Q_1 sorozatok felcserélésével kapott állapotot tükrözi.

A (d) ábrarész a P_1 és P_2, valamint a Q_1 és Q_2 sorozatok (rekurzív) összefésülésével kapott sorozatokat tartalmazza. A következő két ábrarész az összekeveréssel kapott sorozatot, illetve a cserélő-összehasonlítással kapott végeredményt ábrázolja. ♠



3.15. ábra. Páratlan-páros összefésülés láncon

3.5.3. Páratlan-páros összefésülés négyzeten

Ebben a szakaszban négyzet a számítási modell. Feltesszük, hogy a 2 hatványa és a bemenő adatok két részre osztva, a 3.16. ábra (a) részének megfelelően *kígyószerűen* (♣) helyezkednek el az $1,\ 2, \ldots,\ \frac{a}{2}$, illetve az $\frac{a}{2}+1,\ \frac{a}{2}+2,\ \ldots,\ a$ oszlopindexű processzorok helyi memóriájában. Mindkét rész a oszlopból és $\frac{a}{2}$ sorból álló *adatkígyó* (♣).

Ennek a két rendezett sorozatnak az összefésülésére alkalmas a következő algoritmus.

N´ - - ´ ¨ (p) *párhuzamos rekurzív eljárás*

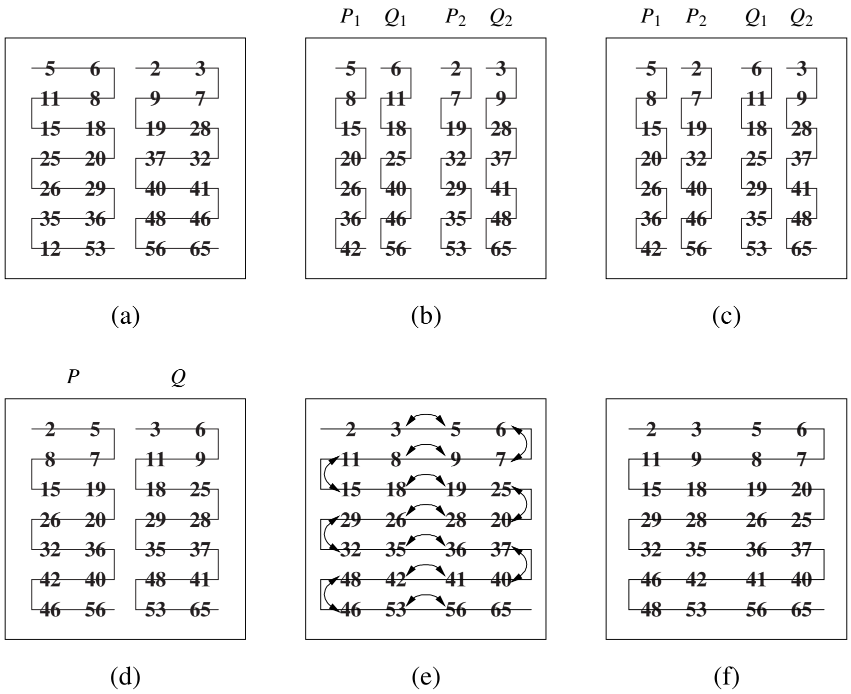
Számítási modell: négyzet

Bemenet: p (processzorszám, 2 páratlan hatványa), X_1 és X_2

(mindkettő $\frac{a}{2}$ hosszúságú rendezett sorozat)

Kimenet: Y (p hosszúságú rendezett sorozat

01 **if** $l = 0$ **then**



3.16. ábra. Páratlan-páros összefésülés négyzeten

```
            Fésüljük össze a két kígyót
        else
            Cseréljük fel P2-t és Q1-et
    Rekurzívan fésüljük össze P1-et és P2-t
```

Most megmutatjuk, hogy ez az algoritmus $O(a)$ lépésben befejeződik.

3.24. tétel. (*Kígyók rendezése rácson.*) *A* R´ - -F´ ¨ *algoritmus két* $a \times \frac{a}{2}$ *hosszúságú rendezett adatkígyót* $O(a)$ *lépésben összefésül egy* $a \times a$ *méretű négyzeten.*

Bizonyítás. Az algoritmus lépésszámára az

$$M(l) \leq M\left(\frac{l}{2}\right) + 2l \tag{3.8}$$

adódik, amelynek a megoldása $M(l) = O(\sqrt{p})$. ■

3.6. Rendezés

A 2. fejezetben foglalkoztunk PRAM modellen rendező algoritmusokkal. Most lánc és rács lesznek a felhasznált számítási modellek.

3.6.1. Rendezés láncon

A L´ - - algoritmus először meghatározza a rendezendő kulcsok rangját, majd eljuttatja a kulcsokat a megfelelő helyre. A L´ - ´ - algoritmus a soros buborékrendezéshez hasonlít.

3.6.1.1. Rangsoroló rendezés láncon

Ez az algoritmus $O(p)$ lépést tesz.

3.25. tétel. *A* L´ - - *algoritmus p kulcsot egy láncon $O(p)$ lépésben rendez.*

3.6.1.2. Páratlan-páros felcserélő rendezés láncon

A L´ - - algoritmus alapötlete ugyanaz, mint a soros buborékrendezésé: a szomszédos kulcsokat összehasonlítjuk és szükség esetén felcseréljük.

L´ - - (p) *párhuzamos eljárás*

Számítási modell: lánc

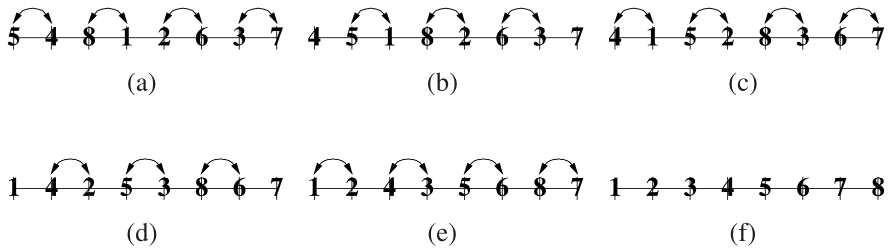
Bemenet: X (rendezendő kulcsok)

Kimenet: Y (p hosszúságú rendezett sorozat)

P_i **in parallel for** $1 \le i \le p$
    if i páratlan **then**
        Összehasonlít és cserél
    else
        Összehasonlít és cserél

Ez az algoritmus is $O(p)$ lépést tesz.

3.26. tétel. *A* L´ - - *algoritmus egy p processzoros láncon p kulcsot $O(p)$ lépésben rendez.*



3.17. ábra. Páros-páratlan felcserélő rendezés láncon

3.6.1.3. Páratlan-páros összefésülő rendezés láncon

A L´ - ´ ¨ - algoritmus alapötlete ugyanaz, mint a soros buborékrendezésé: a szomszédos kulcsokat összehasonlítjuk és szükség esetén felcseréljük.

Ez az algoritmus is $O(p)$ lépést tesz.

3.27. tétel. *A* L´ - ´ ¨ - *algoritmus egy p processzoros láncon p kulcsot $O(p)$ lépésben rendez.*

3.6.2. Rendezés négyzeten

Két algoritmust vizsgálunk ebben az alfejezetben. A S - $\Theta(a \lg a)$ lépést tesz, a másik viszont $O(a)$ lépésszámának köszönhetően munkahatékony.

3.6.2.1. Schearson rendező algoritmusa

S - (p) *párhuzamos eljárás*

Számítási modell: négyzet

Bemenet: X (rendezendő kulcsok)

Kimenet: Y (p hosszúságú rendezett sorozat)

```
for i := 1 to n
    if i páros then
            Rendezzük az oszlopokat
            Rendezzük az első sort növekvőleg
```

3.28. példa. Schearson-rendezés

A 3.18. ábra (a) része egy 4×4 méretű négyzeten elrendezett kulcsokat ábrázol. Az első fázisban a sorokat rendezzük – az egymást követő sorokat ellenkező módon (az első sort növekvőleg, a másodikat csökkenőleg stb.) Az első fázis eredményét mutatja az ábra (b) része. Az ábra következő részei rendre a második, ..., ötödik fázis utáni helyzetet mutatják. Az ötödik fázis végén a négyzet rendezve van. ♠

3.6.2.2. Páratlan-páros összefésülő rendezés

Most a páratlan-páros rendezési algoritmust négyzeten valósítjuk meg. Egy példát mutat a 3.19. ábra.

Az algoritmus lépésszáma a következő.

15	12	8	32
7	13	6	17
2	16	19	25
18	11	5	3

(a)

8	12	15	32
17	13	7	6
2	16	19	25
18	11	5	3

(b)

2	11	5	3
8	12	7	6
17	13	15	25
18	16	19	32

(c)

2	3	5	11
12	8	7	6
13	15	17	25
32	19	18	16

(d)

2	3	5	6
12	8	7	11
13	15	17	16
32	19	18	25

(e)

2	3	5	6
12	11	8	7
13	15	16	17
32	25	19	18

(f)

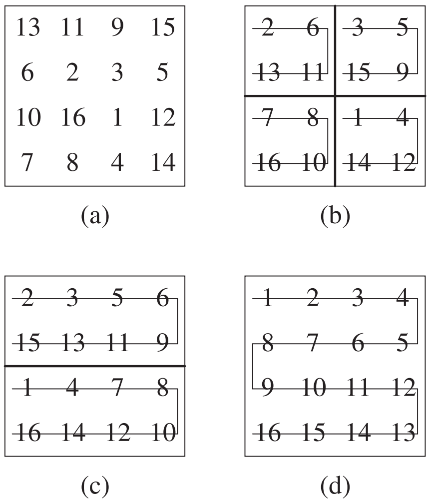
3.18. ábra. Példa Schearson-rendezésre

3.29. tétel. (*Páratlan-páros összefésülő rendezés.*) *p elem* $\sqrt{p} \times \sqrt{p}$ *méretű négyzeten* $O(\sqrt{p})$ *lépésben rendezhető.*

3.30. példa. 16 elem rendezése páratlan-páros összefésüléssel A 3.19. ábra mutatja 16 szám rendezését négyzeten kígyószerű sorfolytonos sorrendbe. ♠

3.7. Gráfalgoritmusok

Ebben az alfejezetben először néhány kockán futó gráfalgoritmust mutatunk be, majd négyzeten oldunk meg gráfokkal kapcsolatos feladatokat.



3.19. ábra. Páratlan-páros összefésülés négyzeten

3.7.1. Kocka

Ebben a szakaszban kocka lesz a számítási modell. A minmátrix, tranzitív lezárt, és az összefüggő komponensek számítását mutatjuk be.

Újra alkalmazzuk a második fejezetben a tranzitív lezárt, összefüggő komponensek és legrövidebb utak meghatározására kidolgozott formalizmust.

Először a minmátrix számítására mutatunk egy algoritmust, amely $n \times n \times n$ méretű kockán $O(n \lg n)$ lépést tesz. Ezután a tranzitív lezárt, a legrövidebb utak és a konvex burok meghatározására mutatunk be négyzeten működő algoritmusokat.

3.7.1.1. Minmátrix számítása

Az M mátrixból az $\overline{M}$ mátrixot egy kockán $O(n \lg n)$ lépésben meg tudjuk határozni.

Egy $n \times n \times n$ méretű *kocka elemeit* (♣) úgy definiáljuk, hogy $(i, *, *)$ azokat a processzorokat jelöli, amelyeknek az első koordinátája i.

3.31. tétel. (*Minmátrix számítása kockán.) Egy $n \times n$ méretű mátrix minmátrixa egy $n \times n \times n$ méretű kockán $O(n \lg n)$ lépésben kiszámítható.*

3.7.1.2. Irányított gráf tranzitív lezártja

Az előzőek alapján adódik a K - ´ algoritmus, melynek lépésszáma a következő.

3.32. tétel. (*Tranzitív lezárt számítása kockán.*) *Egy n csúcsú irányított gráf tranzitív lezárt mátrixa egy $n \times n \times n$ méretű kockán $O(n \lg n)$ lépésben kiszámítható.*

3.7.1.3. Összefüggő komponensek meghatározása

3.33. tétel. (*Összefüggő komponensek számítása rácson.*) *Egy n csúcsú irányított gráf összefüggő komponensei egy $n \times n \times n$ méretű kockán $O(n \lg n)$ lépésben kiszámíthatók.*

Bizonyítás. A minmátrixra vonatkozó tétel segítségével. ■

3.7.2. Négyzet

Ebben az alfejezetben a tranzitív lezárt és a legrövidebb utak számítására determinisztikus, a teljes párosítás keresésére (páros gráfban és általános gráfban) pedig véletlenített algoritmust mutatunk be.

3.7.2.1. Tranzitív lezárt

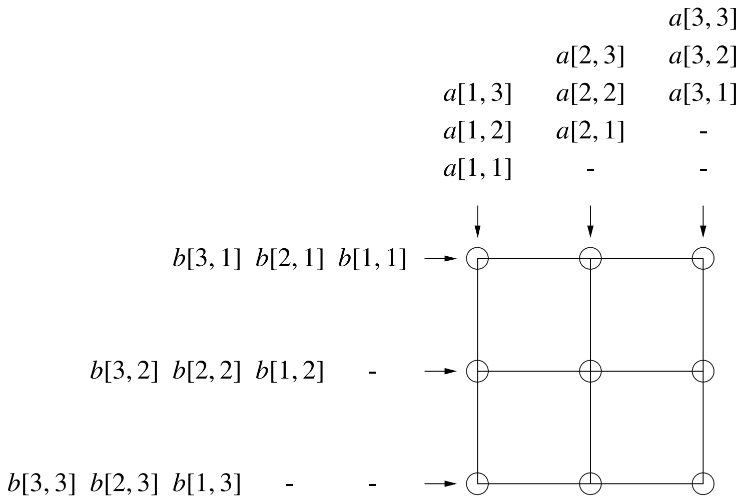
Egy n-csúcsú gráf tranzitív lezártja meghatározható úgy, hogy $n \times n$ méretű mátrixokkal $\lceil \lg n \rceil$ szorzást végzünk. A következő tétel ezen szorzások négyzeten való gyors elvégezhetőségét mondja ki.

3.34. tétel. (*Mátrixok szorzása négyzeten.*) *Az $n \times n$ méretű $A = a[i, j]$ és $B[i, j]$ mátrixok egy $n \times n$ méretű négyzeten $O(n)$ idő alatt összeszorozhatók.*

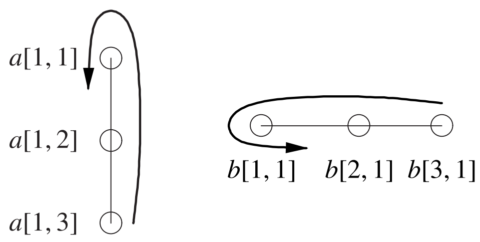
Ennek a tételnek a segítségével belátható a következő állítás.

3.35. tétel. (*Tranzitív lezárt rácson.*) *Adott irányítatlan gráf tranzitív lezárt mátrixa egy $n \times n$ méretű négyzeten $O(n \lg n)$ lépésben meghatározható.*

Bizonyítás. A tétel bizonyítását a 3.20. ábra és a 3.21. ábra segítségével szemléltetjük. ■



3.20. ábra. Két mátrix szorzása



3.21. ábra. Az adatáram szimulálása

3.7.2.2. Legrövidebb utak

A legrövidebb utak minden csúcspárra való meghatározására vonatkozik a következő tétel.

3.36. tétel. (*Legrövidebb utak meghatározása négyzeten.*) *A legrövidebb utak egy gráf minden csúcspárjára egy négyzeten $O(n \lg n)$ lépésben meghatározhatók.*

3.7.2.3. Konvex burok

n síkbeli pont konvex burka egy n-processzoros láncon $O(n)$ idő alatt meghatározható (ld. 3.10. gyakorlat). Ez az idő nagy valószínűséggel lényegesen csökkenthető.

Legyen H_1 és H_2 a síkbeli pontok két felső burka.

3.37. lemma. (*Érintő számítása.*) *Legyen H_1 és H_2 két olyan felső burok, amelynek legfeljebb n pontja van. Ha P pontja H_1-nek, akkor a H_2-vel bezárt szöge $O(\sqrt{n})$ lépésben meghatározható.*

3.38. lemma. (*Két felső burok közös érintője.*) *Ha H_1 és H_2 két felső burok legfeljebb n ponttal, akkor közös érintőjük $O(\sqrt{n} \lg n)$ lépésben meghatározható.*

A két lemmából adódik a következő tétel.

3.39. tétel. (*Konvex burok számítása.*) *n síkbeli pont közös konvex burka egy $\sqrt{n} \times \sqrt{n}$ méretű négyzeten $O(\sqrt{n} \lg^2 n)$ lépésben meghatározható.*

3.40. példa. 16 pont konvex burka

A 3.22. ábra (a) része 16 pont adatait mutatja. Ezek az adatok egy 4×4 méretű négyzetnek megfelelően vannak elrendezve. A 4 részre tagolt adatok minden negyedrészben az x-koordináták szerint rendezve tartalmazzák az adatokat (azonban nem sorfolytonosan, hanem kígyószerű sorrendben). A felső burok (rekurzív) kiszámításának eredményét mutatja az ábra (b) része. A két felső, és a két alsó negyedrész összefésülésének eredményét mutatja az ábra (c) része. Végül az ábra felső részén, illetve alsó részén ábrázolt burok összefésülésével kapjuk az ábra (d) részén bemutatott végeredményt. ♠

Az összefésülés gyorsabb megoldásával csökkenthetjük az algoritmus lépésszámát. A F ˝- - ´ ¨ algoritmus összesen legfeljebb a^2 pontot tartalmazó felső burkokat $O(a)$ lépésben összefésül.

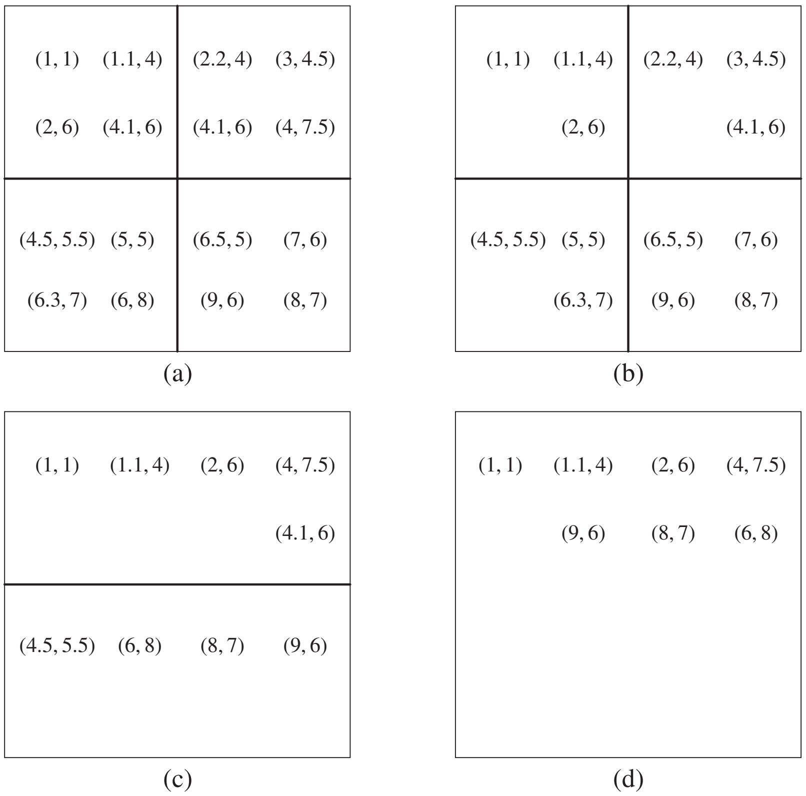
F ˝- - ´ ¨ (a, u, v) *párhuzamos rekurzív eljárás*

Számítási modell: négyzetrács

Bemenet: B_1 és B_2 (mindkettő legfeljebb $\frac{a^2}{2}$ pontot tartalmazó felső burok)

Kimenet: Y (p hosszúságú rendezett sorozat)

if $l = 1$ **then**
legyen u a baloldali pont és v a jobboldali pont
Legyen p a H_1 középső pontja. Keressük meg a p-ből H_2-höz
húzott érintő H_2-vel közös q pontját. Állapítsuk meg u és p ...
Hagyjuk el H_1-nek azt a felét, amely nem tartalmazza u-t.
Hasonlóképpen hagyjuk el H_2 felét is.



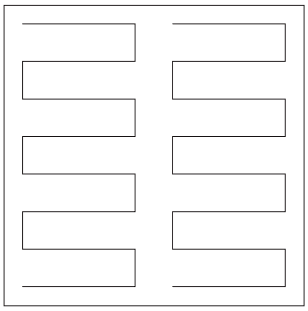
3.22. ábra. Felső burok számítása

Ismételjük ezt a lépést addig, amíg H_1 és H_2 negyedrésze marad

Rendezzük át H_1 és H_2 megmaradó pontjait úgy, hogy az $l/2 \times l/2$ méretű részrácsot a 3.23. ábra szerint foglalják el

Rekurzívan dolgozzunk a négyzeten.

3.41. lemma. *A* F ˝- - ´ ¨ *algoritmus egy $a \times a$ méretű rácson $\overline{O}(a)$ lépésben összefésül két konvex burkot.*



3.23. ábra. Két felső burok összefésülése

Innen adódik, hogy a konvex burkot nagy valószínűséggel az előbbinél gyorsabban is meg tudjuk határozni.

3.42. tétel. (*Konvex burok négyzeten.*) *A* K - - ´ *algoritmus egy $a \times a$ méretű négyzeten $\overline{O}(a)$ lépésben meghatározza a^2 pont konvex burkát.*

3.8. Gyakorlatok

3-1. Mennyi a FOF és a LIFO elsőbbségi algoritmusok lépésszáma a 3.1. példa adatai esetében?

3-2. Egy p processzoros lánc minden processzoránál két csomag van. Feltesszük, hogy p páros. A P_i processzornál lévő csomagok rendeltetési helye a $P_{p/2+i}$ ($i = 1, 2, \ldots, \frac{p}{2} + i$). Minden csomag a számára legrövidebb úton jut el a célba. Határozzuk meg a csomagok utazási idejét a FOF, FDF, FIFO és LIFO elsőbbségi módszerek esetében.

3-3. Egy $a \times a$ méretű rácsban minden processzortól legfeljebb $k \geq 1$ csomag indul és minden processzorhoz legfeljebb k csomag érkezik. Tervezzünk algoritmust, amely legfeljebb $\frac{(k+1)p}{2}$ lépésben megoldja a csomagirányítási feladatot.

3-4. Mutassuk meg, hogy egy p processzoros gyűrűben a PPR probléma $\frac{p}{2}$ lépésben megoldható.

3-5. Hogyan oldható meg a szuffixszámítási feladat egy $\sqrt{p} \times \sqrt{p}$ méretű rácson $O(\sqrt{p})$ lépésben?

3-6. p elem közül kell a maximálisat megkeresni egy $\sqrt{p} \times \sqrt{p}$ méretű rács segítségével. Az $\mathcal{A}$ algoritmus $T(\sqrt{p})$ lépésben meghatározza p elem súlyozott mediánját. Hogyan használható fel $\mathcal{A}$ a keresésre és milyen lépésszámot kapunk?

3-7. Legyen

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0. \tag{3.9}$$

Tervezzünk algoritmust, amely láncon és rácson meghatározza az $f(x)$ polinom értékét egy $x = x_0$ helyen. Mennyi lesz a tervezett algoritmus lépésszáma?

3-8. Tekintsünk egy $\sqrt{p} \times \sqrt{p}$ méretű négyzetet, melynek minden processzorának memóriájában van egy, a $[0,\ p^{\epsilon}]$ intervallumból vett egész szám, ahol ϵ a $(0, 1)$ intervallumból vett állandó. Tervezzünk algoritmust a legnagyobb kulcs kiválasztására. Mennyi lesz e tervezett algoritmus lépésszáma?

3-9. Valósítsuk meg a rangsorolási algoritmust egy $\sqrt{p} \times \sqrt{p}$ méretű rácson.

3-10. Mutassuk meg, hogy p pont konvex burka $O(p)$ lépésben meghatározható egy p processzoros láncon.

3-11. Tervezzünk algoritmust, amely láncon és rácson meghatározza, hogy adott n pont között van-e 3 olyan pont, amelyek egy egyenesre esnek. Mit mondhatunk algoritmusaink lépésszámáról és a felhasznált processzorok számáról?

3.9. Feladatok

3-1. Csomagok irányítása közeli célokhoz

Egy $a \times a$ méretű rácsban minden processzor legfeljebb egy üzenetet küld és legfeljebb d távolságra. Tervezzünk csomagirányítási algoritmust, amely $\overline{O}(d)$ lépést tesz.

3-2. Csomagirányítás tóruszon

Módosítsuk úgy a H´ - ´ ´ algoritmust, hogy tóruszon $1.5\sqrt{p} + \overline{o}(p)$ lépést tegyen.

3-3. Palindrómák ellenőrzése

Egy adott Σ ábécé feletti $w = x_1 x_2 \ldots x_p$ szót akkor nevezünk *palindrómának*, ha $w = x_p x_{p-1} \ldots x_1$ teljesül. Tervezzünk párhuzamos algoritmust, amely egy p processzoros láncon $O(p)$ lépésben eldönti, hogy egy p hosszúságú szó palindróma-e.

3-4. Gyors Fourier-transzformált kiszámítása

Tervezzünk algoritmust, amely egy p processzoros láncon $O(p)$ lépésben kiszámítja egy p hosszúságú vektor FFT-jét (gyors Fourier-transzformáltját). Tervezzünk rácsot használó algoritmust az FFT kiszámítására. Milyen lépésszám érhető el?

3-5. Egycsomagos csomagirányítás

Tegyük fel, hogy egy $\sqrt{p} \times \sqrt{p}$ méretű rácsban minden processzor pontosan egy csomagot küld és pontosan egy csomagot fogad. Tervezzünk olyan csomagirányító algoritmust, melynek lépésszáma $O(\sqrt{p})$, az igényelt sorhosszúsága pedig $O(1)$.

3-6. Csoportokba rendezés láncon

Tegyük fel, hogy egy $\lg n$ processzoros lánc minden processzorának memóriájában $\frac{n}{\lg n}$ kulcs van. Tervezzünk algoritmust, amely biztosítja, hogy P_1 memóriájába kerüljön a legkisebb $\frac{n}{\lg n}$ kulcs, P_2 memóriájába a következő $\frac{n}{\lg n}$ legkisebb kulcs, és így tovább, a legnagyobb indexű processzor memóriájába kerüljön a $\frac{n}{\lg n}$

legnagyobb kulcs. Mutassuk meg, hogy ez a *rendezés* megoldható $O(n)$ lépésben.

3-7. *Soronkénti és oszloponkénti rendezés*

Mutassuk meg, hogy ha egy $\sqrt{p} \times \sqrt{p}$ méretű rácsban minden processzor memóriájában van egy kulcs és ezeket a kulcsokat először soronként, azután oszloponként rendezzük, akkor a sorok és oszlopok is rendezettek lesznek.

3-8. *Prefix számítása bináris fával*

Processzorok bináris fája (röviden: bináris fa) olyan teljes bináris fa, melynek minden csúcsában van egy processzor és az élek adatátviteli vonalak. A 4.8. ábra egy 4 levelű bináris fát mutat. A bemenő adatok rendszerint a fa levelein jelennek meg. A n levelű bináris fában $2n - 1$ processzor van és a fa magassága $\lceil \lg n \rceil$. Ha minden levélen van egy szám, ezen számok összegét kiszámíthatjuk a következőképpen. Először minden levél elküldi a nála lévő számot a szülőjének. Ha egy belső processzor kap két számot alulról, akkor összeadja őket és az összeget elküldi a szülőjének. Ily módon $\lg n$ lépés után a gyökérben megjelenik az összeg.

Oldjuk meg a prefixszámítási problémát egy n levelű bináris fával. Kezdetben minden levélnél van egy elem. A prefixek értékét a levelekről lehet kivinni. Hogyan oldható meg a feladat $O(n)$ lépésben?

3-9. *Topologikus rendezés rácson*

Tervezzünk algoritmust, amely rács segítségével topologikusan rendez.

3-10. *Körmentesség ellenőrzése*

Tervezzünk algoritmust, amely rácson ellenőrzi, hogy adott irányítatlan gráf tartalmaz-e kört?

3-11. *Minimális feszítőfa 0-1 súlyok esetében*

Tegyük fel, hogy az irányított gráfok éleinek súlya nulla vagy egy lehet. Tervezzünk rácsalgoritmust, amely meghatározza a minimális feszítőfát.

3-12. *Háromszög mátrix invertálása négyzeten*

Mutassuk meg, hogyan lehet egy $a \times a$ méretű háromszög mátrixot $O(a)$ lépésben invertálni egy $a \times a$ méretű négyzeten.

3-13. Háromátlós mátrix invertálása négyzeten

Mutassuk meg, hogyan lehet egy $a \times a$ méretű háromátlós mátrixot $O(a)$ lépésben invertálni egy $a \times a$ méretű négyzeten.

3-14. Konvex burok területe

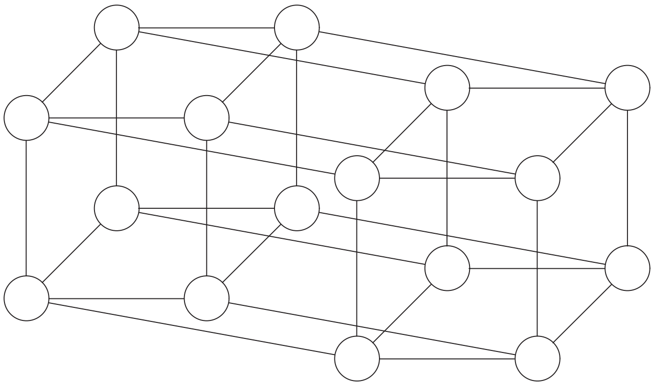
Tervezzünk olyan algoritmust, amely egy $a \times a$ méretű négyzeten $\overline{O}(a)$ lépésben meghatározza a^2 pont konvex burkának területét.

4. Hiperkocka

4.1. Számítási modellek

4.1.1. Hiperkocka

Az első fejezetben szereplő definíció szerint egy d-dimenziós hiperkocka, amelyet $\mathcal{H}_d$-vel jelölünk, 2^d processzorból áll. $\mathcal{H}_d$ minden processzora megcímkézhető egy d bites bináris számmal. A harmadik fejezetben lévő 3.3. ábra egy 3-dimenziós, a 4.1. ábra pedig egy 4-dimenziós hiperkockát ábrázol.



4.1. ábra. 4-dimenziós hiperkocka

A processzort és címkéjét ugyanazon szimbólummal jelöljük.

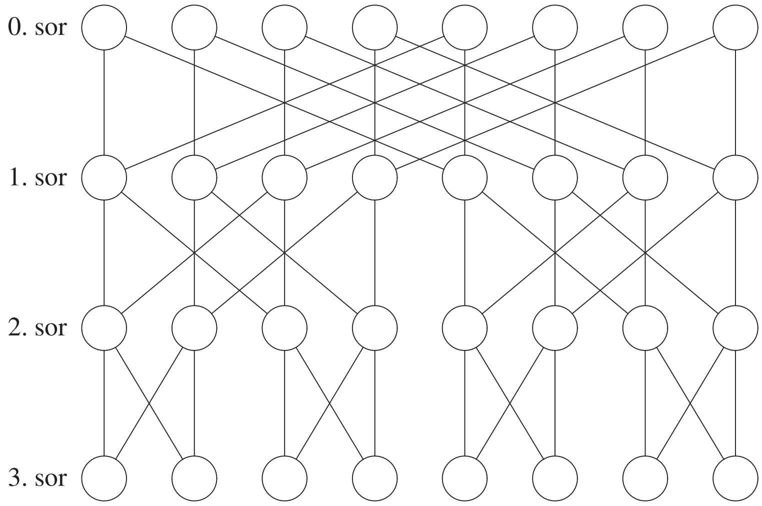
Ha v egy d bites bináris szám, akkor v első bitjét tekintjük legmagasabb helyiért ékűnek. Jelölje $v(i)$ azt a d bites bináris számot, amely v-től csak az i-edik bitjében tér el. $\mathcal{H}_d$ minden P_v processzorára igaz, hogy az pontosan a $P_{v(i)}$ $(i = 1, 2, \ldots, d)$ processzorokkal van összekapcsolva. A $(v, v(i))$ kapcsolatot *i-edik szintű kapcsolatnak* nevezzük. Mivel $\mathcal{H}_d$ minden processzora pontosan d másikkal van összekötve, így $\mathcal{H}_d$ d-reguláris és a fokszáma d.

Az u és v bináris számok Hamming-távolsága azon bitpozíciók száma, amelyeken a két bináris szám eltér. Jele $H(u, v)$. A $\mathcal{H}_d$ hiperkocka bármely P_u és P_v processzora között van $H(u, v)$ hosszúságú út. Ha ugyanis u és v két processzor $\mathcal{H}_d$-ben, akkor egy közöttük vezető út megadható a következő módon. Legyenek $i_1, i_2, \ldots, i_k$ azon bitpozíciók (növekvő sorrendben) amelyeken P_u és P_v eltérnek. Ekkor létezik a következő útvonal: $u = w_0, w_1, w_2, \ldots, w_k = v$, ahol $w_j = w_{j-1}(i, j)$ $(1 \leq j \leq k)$. Ebből következik, hogy egy d-dimenziós hiperkocka átmérője pontosan d. A hiperkocka minden processzora egy helyi memóriával rendelkező RAM, amely minden alapvető műveletet (összeadás, kivonás, szorzás, osztás, összehasonlítás vagy a hozzáférés a helyi memóriához stb.) egységnyi idő alatt végez el. A processzorok közötti kommunikáció a processzorokat összekötő kapcsolatok mentén történik. Ha két processzor között nincs közvetlen kapcsolat, akkor a kommunikáció az egyik processzortól a másikig vezető út mentén lehetséges, ám az adatátvitel időigénye egyenesen arányos az út hosszával.

A kommunikáció egyidejűsége szempontjából kétféle hiperkockát különböztethetünk meg. Az első típus a *soros* (♣) vagy egyportos hiperkocka, amelynél egy processzor egységnyi idő alatt egyetlen szomszédjával képes kommunikálni. Ezzel szemben a *párhuzamos* (♣) vagy többportos hiperkocka egy processzora egységnyi idő alatt mind a d szomszédjával képes adatot cserélni. A továbbiakban mindig jelölni fogjuk, hogy melyik kommunikációs modellt használjuk. Mindkét megoldás szinkron processzorműködést tételez fel, azaz minden időegységben a processzorok mindegyike pontosan egy utasítást hajt végre. A hiperkockáknak több – számunkra kedvező – tulajdonsága is van. Az egyik a kicsi átmérő. Egy p-processzoros hiperkockának az átmérője $\lg p$, míg az azonos számú processzort tartalmazó rács átmérője legalább $2(\sqrt{p} - 1)$. Egy másik kedvező tulajdonság, hogy $\mathcal{H}_{d+1}$ felépíthető rekurzívan. Vegyük ugyanis $\mathcal{H}_d$ két példányát, $\mathcal{H}'$-t és $\mathcal{H}''$-t. Egészítsük ki $\mathcal{H}'$ processzorainak címkéjét a 0 prefixszel, $\mathcal{H}''$-ét pedig az 1 prefixszel. Ezután $\mathcal{H}'$ minden P_v processzorát kössünk össze $\mathcal{H}''$ $P_{v(1)}$ processzorával. Ez a tulajdonság megfordítva azt is jelenti, hogy $\mathcal{H}_{d+1}$ $\mathcal{H}_d$-nek két példányát tartalmazza. Például azon processzorok, amelyek címkéjének első bitje 0, ha csak a közöttük futó kapcsolatokat vesszük figyelembe, pontosan kiadják $\mathcal{H}_d$-t. Sőt, tetszőleges $1 \leq q \leq d$-re, azon processzorok, amelyek címkéjének q-adik bitje azonos, kiadják $\mathcal{H}_d$-t. Még általánosabban, ha rögzítjük i $(1 \leq i \leq d+1)$ bit értékét, a megadott feltételt kielégítő processzorok $\mathcal{H}_{(d+1)-i}$-t alkotnak.

4.1.2. Pillangó hálózat

A pillangóhálózat közeli kapcsolatban áll a hiperkockákkal. A pillangóhálózatokra tervezett algoritmusok könnyedén átültethetők hiperkockákra és fordítva. Valójában gyakran könnyebb az adott probléma megoldását pillangóhálózaton elkészíteni, majd onnan átültetni hiperkockára. Egy d-dimenziós pillangóhálózat, amelyet $\mathcal{B}_d$-vel fogunk jelölni, $p = (d + 1)2^d$ processzorból és $d2^{d+1}$ élből áll. $\mathcal{B}_d$ minden processzora egy $\langle r, l\rangle$ párral jellemezhető, ahol $0 \leq r \leq 2^d - 1$ és $0 \leq l \leq d$. Az r változót a processzor *sorindexének* (♣) nevezzük, míg l a processzor *szintje*. Egy $P_u = \langle r, l\rangle$ processzor $(0 \leq l < d)$ $\mathcal{B}_d$-ben két, az $(l+1)$-edik szinten levő processzorral van összekötve, a $P_v = \langle r, l+1\rangle$ és $w = \langle r(l+1), l+1\rangle$ processzorokkal. Az (u, v) kapcsolatot *közvetlen kapcsolatnak* (♣), (u, w)-t pedig *kereszt kapcsolatnak* (♣) nevezzük. Mindkét típust $(l + 1)$-*edik szintű* kapcsolatnak mondjuk. Mivel minden processzor pontosan négy másikkal van összekötve, így $\mathcal{B}_d$ csúcsainak fokszáma (d-től, illetve p-től függetlenül) négy, ezért a d-dimenziós pillangó hálózat 4-reguláris és a fokszáma 4. Ha P_u egy 0-adik szintű processzor és P_v egy d-edik szintű, akkor létezik (és egyértelműen meg van határozva) egy d hosszúságú út P_u és P_v között. Legyen $P_u = \langle r, 0\rangle$ és $P_v = \langle r', d\rangle$. Ekkor az út $\langle r, 0\rangle, \langle r_1, 1\rangle, \langle r_2, 2\rangle, \ldots, \langle r', d\rangle$, ahol r_i-nek az első i bitje megegyezik r'-vel, a többi pedig r-rel $(1 \le i \le d-1)$. Vegyük észre, hogy ez az út létezik a pillangó-kapcsolatok definíciója miatt. Ezt az utat *mohó útnak* (♣) nevezzük. A 4.2. ábrán $\mathcal{B}_3$ látható. A vastagított vonal a $P_u = \langle 100, 0 >$ és $P_v = \langle 010, 3\rangle$ közötti mohó utat jelöli.



4.2. ábra. 4 sorban 32 processzort tartalmazó pillangó hálózat

A fentiekből következik, hogy $\mathcal{B}_d$-ben bármely két processzor távolsága legfeljebb $2d$. A korlát éles, hiszen például a $P_u = \langle 0, d\rangle$ és $P_v = \langle 2d-1, d\rangle$ processzorok távolsága pontosan ennyi, így $\mathcal{B}_d$ átmérője $2d$. A pillangóhálózat is rendelkezik a hiperkockához hasonló rekurzív tulajdonsággal. Ha $\mathcal{B}_d$-ből eltávolítjuk a 0-adik szinten lévő processzorokat a hozzájuk csatlakozó élekkel együtt, akkor a hálózat két $(d-1)$-dimenziós pillangóhálózatra esik szét.

$\mathcal{H}_d$ és $\mathcal{B}_d$ között szoros kapcsolat van. Ha $\mathcal{B}_d$ minden sorát egyetlen processzorba fogjuk össze, megtartva a kimenő éleket, akkor az eredményül kapott gráf $\mathcal{H}_d$ (a keletkező többszörös éleket egyetlen példánnyal helyettesítve). Ennek következményeként kapjuk a következő lemmát.

4.1. lemma. (*Pillangó hálózat szimulálása.*) *$\mathcal{B}_d$ minden végrehajtási lépése szimulálható egy párhuzamos $\mathcal{H}_d$ egyetlen lépésével vagy egy soros $\mathcal{H}_d$ d lépésével.*

Egy $\mathcal{B}_d$ algoritmust *normálisnak* (♣) nevezünk, ha minden időpillanatban legfeljebb egy szint processzorai vesznek részt a kiszámításban.

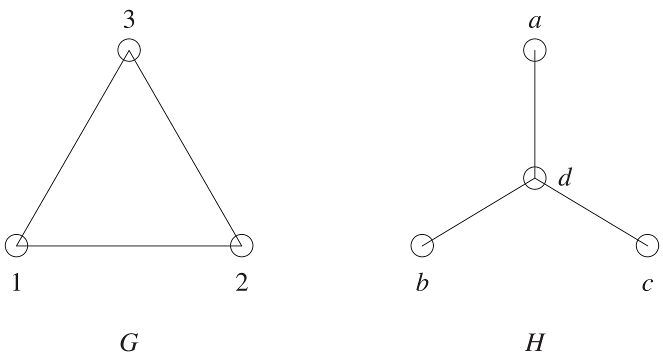
4.2. lemma. (*Normális algoritmus szimulálása.*) *Egy normális $\mathcal{B}_d$ egy lépése szimulálható egy soros $\mathcal{H}_d$ egyetlen lépésével.*

4.1.3. Hálózatok beágyazása

Egy hálózatnak egy másikra történő leképezését *beágyazásnak* nevezzük. Formálisan, a $G(V_1,E_1)$ hálózat beágyazása a $H(V_2,E_2)$ hálózatba, egy leképezés V_1-ről V_2-re. G-nek H-ra való leképezését felhasználva a G hálózatra tervezett algoritmusok lefuttathatók a H hálózaton. A 4.3. ábra bal oldalán látható G hálózat egy lehetséges beágyazása H-ba a következő leképezés: $1 \to b,\ 2 \to c,\ 3 \to a$.

A beágyazás *felfúvódásának* (♣) a $\frac{|V_2|}{|V_1|}$ hányadost nevezzük. A beágyazás *késleltetése* (♣) a leghosszabb út hossza, amelyre G-nek egy éle leképződött. A H hálózat egy éle *torlódásának* (♣) nevezzük az adott élt használó olyan utak számát, amelyekre G valamely élét leképeztük. A *beágyazás torlódása* (♣) a H élei torlódásának maximuma.

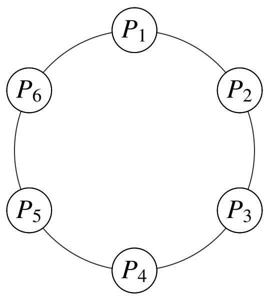
A 4.3. ábrán látható példában a felfúvódás mértéke $\frac{4}{3}$, a késleltetés 2, mivel minden él egy-egy kettő hosszúságú útra képződött le. Hasonlóan a H összes élének torlódása 2, így az egész leképezés torlódása is 2.



4.3. ábra. Példa beágyazásra

4.1.3.1. Gyűrű beágyazása

A p-processzoros gyűrű egy síkhálózat, amelyben a P_i $(1 \leq i \leq p)$ processzor a P_{i+1} és a P_{i-1} processzorokkal van összekötve – ahol az indexeket (mod p) vesszük. A 4.4. ábra egy 6-processzoros gyűrűt ábrázol.



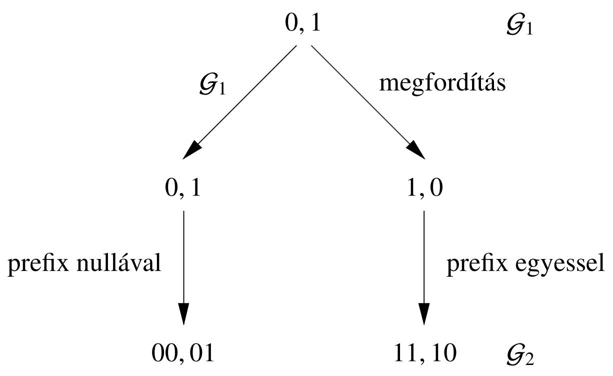
4.4. ábra. 6-processzoros gyűrű

Megmutatjuk hogyan lehet egy 2^d-processzoros gyűrűt beágyazni $\mathcal{H}_d$-be. $\mathcal{H}_d$ processzorait d-bites bináris számokkal címkézzük. Ha a gyűrű processzorait 0-tól $2^d - 1$-ig indexeljük a gyűrű mentén, akkor a 0-s processzor $\mathcal{H}_d$ $000\ldots00$

jelű processzorára fog leképződni, a többi processzor megfelelőjét a Gray-kód segítségével határozhatjuk meg.

A *k-ad rendű Gray-kód* (♣) – jele $\mathcal{G}_k$ – a k-bites bináris számok egy adott permutációját definiálja. Az elsőrendű Gray-kód (G_1) a következő: 0, 1. $\mathcal{G}_k$-t ($k > 1$)-re rekurzívan definiáljuk a következőképpen: $0[\mathcal{G}_{k-1}], 1[\mathcal{G}_{k-1}]R$, ahol $0[\mathcal{G}_{k-1}]$ a $(k-1)$-edrendű Gray-kód elemeit jelenti úgy, hogy mindegyikükhöz hozzáragasztunk egy 0 prefixet. Hasonlóképpen $1[\mathcal{G}_{k-1}]R$ is a $(k-1)$-edrendű Gray-kód elemeit jelenti, ám fordított sorrendben és 1-es prefixszel.

A 4.5. ábra azt mutatja, hogyan származtatható $\mathcal{G}_2$ a $\mathcal{G}_1$ kódból. $\mathcal{G}_0$ elemei a nullás prefixszel a 00, 01 sorozatok adják, $\mathcal{G}_1$ elemeinek megfordítása az egyes prefixszel pedig az 11, 10 sorozatot eredményezi. Tehát $0[\mathcal{G}_2] = 00, 11, 11, 10$.



4.5. ábra. Gray-kód létrehozása

A $\mathcal{G}_k$ i-edik elemét ($0 \le i \le 2^k - 1$) $g(i,k)$-val jelöljük.

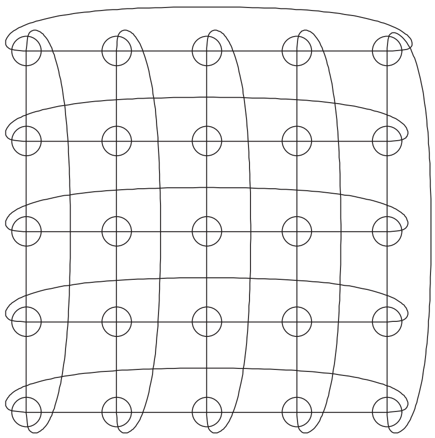
A Gray-kód egyik tulajdonsága, hogy a szomszédos elemek pontosan egy bitben térnek el egymástól. Ebből az következik, hogy $\mathcal{G}_d$ a $\mathcal{H}_d$ processzorainak egy olyan permutációja, hogy a sorban egymást követő processzorok össze vannak kötve éllel a hiperkockában, azaz a gyűrű i-edik processzorát ($0 \le i \le 2^d - 1$) a hiperkocka $g(i,d)$ processzorára képezzük le.

4.3. tétel. (*Gyűrű beágyazása hiperkockába.*) *Egy* 2^d *processzorból álló gyűrű beágyazható* $\mathcal{H}_d$*-be úgy, hogy a felfúvódás, a késleltetés és a torlódás mindegyike* 1.

4.1.3.2. Tórusz beágyazása

Egy $m \times n$ tórusz származtatható egy $m \times n$ méretű rácsból úgy, hogy a rács minden sorának első és utolsó processzorát, valamint minden oszlopának első és utolsó processzorát is összekötjük: tehát a rácsot kiegészítjük a $P_{i,1}$-$P_{i,m}$ $(1 \le i \le m)$ és $P_{1,j}$-$P_{1,n}$ $(1 \le j \le n)$ élekkel.

Már a 3×3 méretű tórusz ábrázolásához is 3 dimenzióra van szükségünk. A 4.6. ábra egy 5×5 méretű tóruszt ábrázol.



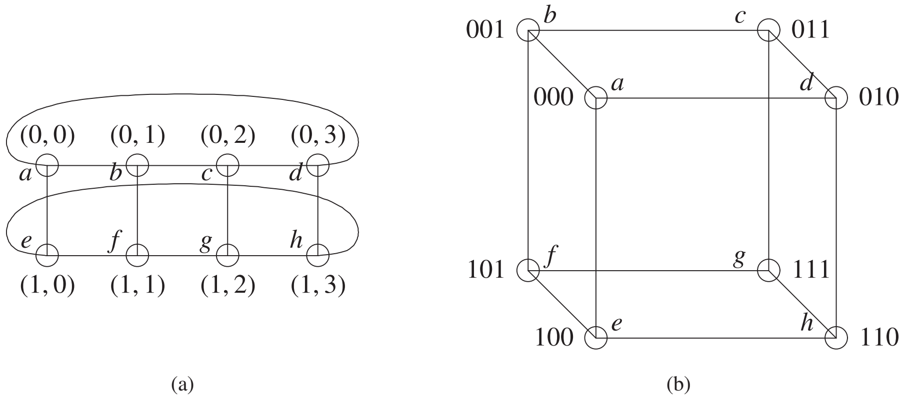
4.6. ábra. 5×5 méretű tórusz

Legyen M egy $2^r \times 2^c$ méretű tórusz. Megmutatjuk, hogy M beágyazható úgy egy hiperkockába, hogy a felfúvódás, a késleltetés és a torlódás mindegyike 1 legyen. Ez az állítás egyszerűen adódik az előző fejezet végén kimondott lemmából. Van 2^r sor és 2^c oszlop M-ben. Korábban már láttuk, hogy ha egy d-dimenziós hiperkockában a d bitből valamely q-t rögzítjük ($1 \le q \le d-1$), akkor a feltételnek megfelelő processzorok egy $\mathcal{H}_{d-q}$ alkockát alkotnak $\mathcal{H}_d$-ben. Ha egy $(r+c)$-bites bináris számnak rögzítjük az r *legnagyobb helyi értékű bitjét* (♣) (MSB = most significant bits) és a maradék c bitet tetszőlegesen variáljuk, a kapott 2^c szám egy alkockát határoz meg $\mathcal{H}_{r+c}$-ben. A fenti lemma értelmében ebbe az alkockába beágyazható egy c processzorból álló gyűrű. Az r MSB minden lehetséges megválasztásához megvan a megfelelő $\mathcal{H}_c$, így M minden sorát leképezhetjük egyre. Egészen pontosan az i-edik sort azon $\mathcal{H}_c$-re képezzük, amelyet úgy kapunk, hogy az r MSB értékét pontosan $g(i,r)$-re állítjuk. Ebből következik, hogy általában a tórusz $P_{i,j}$ processzora a $\mathcal{H}_{r+c}$ $P_{g(i,r),g(j,c)}$ processzorára képződik. Így minden sor szomszédos processzorai a hiperkocka szomszédos processzoraira képződnek. A fentihez hasonló gondolatmenettel belátható, hogy a megadott leképezés az oszlopok szomszédos elemeit is szomszédos processzorokra képezi. Ebből következik, hogy mind a felfúvódás a késleltetés és a torlódás 1.

A 4.7. ábra egy 2×4 méretű tórusznak a $\mathcal{H}_3$ pillangó hálózatba való beágyazását mutatja (a processzoroknak csak az indexe szerepel). Az ábra (a) részén ábrázolt tórusznak 2 sora (a nulladik és az első), valamint négy oszlopa (nulladik, első, második és harmadik) van. Például a tórusz $P_{1,2}$ csúcsát a hiperkocka $P_{g(1,1)g(2,2)} = P_{1,1,1,}$ csúcsára képezzük le. Az ábrán mind a két csúcsot g betűvel jelöltük.

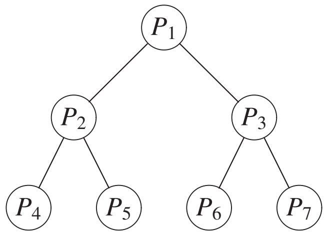
4.1.3.3. Bináris fa beágyazása

Egy d szintes (teljes) bináris fában $p = 2^d - 1$ processzor van: $P_1, P_2, \ldots, P_p$. Az adatszerkezetekkel kapcsolatos terminológia szerint a P_1 processzort *gyökérnek* (♣), a $P_{(p+1)/2}$, $P_{(p+1)/2+1}$, $\ldots$, P_p processzorokat *levélnek* (♣) a többi processzort *belső processzornak* (♣) nevezzük. Ha P_i nem levél, akkor össze van kötve a *gyerekeinek* (♣) nevezett P_{2i} és P_{2i+1} processzorokkal. Ha P_j nem a gyökér, akkor össze van kötve a *szülőjének* nevezett $P_{\lfloor j/2 \rfloor}$ processzorral. A 4.8. ábra



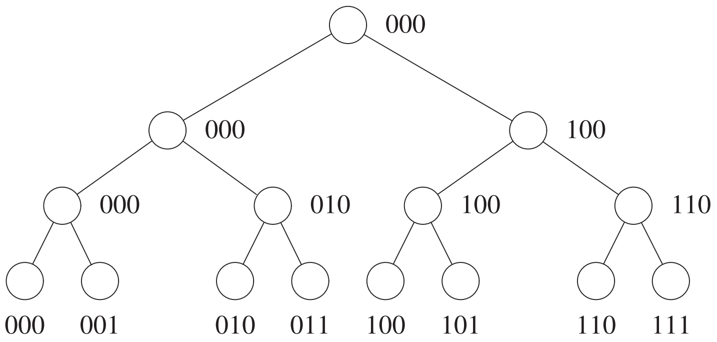
4.7. ábra. Tórusz beágyazása hiperkockába

egy 3 szintes bináris fa hálózatot ábrázol.



4.8. ábra. 3 szintes bináris fa hálózat

Bináris fák sokféle módon beágyazhatók hiperkockába. A következőkben megmutatjuk, hogy egy F p levelű (teljes, bináris) fa (ahol $p = 2^d$) beágyazható $\mathcal{H}_d$-be. Mivel a p levelű fának $2p - 1$ processzora van, így a leképezés nem lehet kölcsönösen egyértelmű. Ha a leveleket $0,\ 1, \ldots, p - 1$ jelöli, akkor képezzük az i-edik levelet $\mathcal{H}_d$ i-edik processzorára. F belső processzorait pedig képezzük arra a processzorra, amelyre az adott processzor legbaloldalibb leszármazottját képeztük. A 4.9. ábrán egy 8 levelű bináris fa csúcsainak leképezését láthatjuk.



4.9. ábra. Bináris fa beágyazása hiperkockába

A fa csúcsai melletti bitsorozat a $\mathcal{H}_3$ hiperkocka megfelelő csúcsának címkéje. Például a hiperkocka 0,0,0 csúcsára a fa 4 csúcsát képeztük le, míg az 1,1,1 csúcsra csak a fa egyetlen csúcsát.

A megadott beágyazás segítségével fa algoritmusokat hatékonyan szimulálhatunk soros hiperkockákkal. Ha a fa algoritmusban a számítás egy lépésében a fának legfeljebb egy szintjén lévő processzorok vesznek részt, akkor az a lépés a hiperkocka egyetlen lépésével szimulálható.

4.2. Csomagirányítás

A probléma: minden processzor legfeljebb egy csomagot küld és minden processzor legfeljebb egy csomag címzettje. Juttassuk el a csomagokat a feladótól a címzettekhez.

4.2.1. Mohó algoritmus

Tekintsük a csomagirányítási problémát először egy pillangóhálózaton, ahol kezdetben minden csomag a nulladik szinten helyezkedik el, a címzett processzorok pedig a d-ediken, azaz a címzett sorok a küldő sorok egy *parciális permutációját* alkotják. A mohó megoldás a csomagirányítási problémára ekkor az, hogy minden csomagot a mohó úton küldünk címzettjéhez. Ekkor minden csomag pontosan d hosszúságú utat tesz meg, így a továbbiakban az algoritmus vizsgálatához csak azt kell megvizsgálnunk, hogy mennyi késleltetést szenvedhet el egy csomag útja során. Legyen $u = \langle r, l \rangle$ egy tetszőleges processzor $\mathcal{B}_d$-ben. Ekkor legfeljebb 2^l csomag mehet keresztül u-n, hiszen a hálózat minden processzorának (a nulladik szintet kivéve) pontosan két szomszédja van a felette levő szinten. Hasonlóan a processzoron átmenő minden csomag címzettje az u-ból elérhető 2^{d-l} d-edik szinten levő processzor valamelyike. Ebből az következik, hogy az u processzorba befutó bármelyik l-edik szintű élen legfeljebb $\min(2^{l-1}, 2^{d-l})$ csomag osztozik. Legyen most p egy tetszőleges csomag. p egy l-edik szintű élen áthaladva legfeljebb $\min(2^{l-1}, 2^{d-l})$ késleltetést szenved el ($l = 1, 2, \ldots, d$). Így a maximális késleltetés, ami összesen egy csomagot érhet:

$$D = \sum_{l=1}^{d} \min\{2^{l-1}, 2^{d-l}\}. \tag{4.1}$$

Az általánosság megszorítása nélkül feltehetjük, hogy d páros. Ekkor D felírható a következőképpen:

$$\begin{aligned} D &= \sum_{l=1}^{d/2} 2^{l-1} + \sum_{l=d/2+1}^{d} 2^{d-l} && (4.2)\\ &= 2 * 2^{d/2} - 2 && (4.3)\\ &= O(2^{d/2}). && (4.4) \end{aligned}$$

Az $O(2^{d/2})$ érték felső korlát a maximális sorhosszúságra, ugyanis – mint láttuk – egy l-edik szintű processzoron legfeljebb $\min(2^{l-1}, 2^{d-l})$ csomag haladhat át. Ezen érték maximuma ($l = 1, 2, \ldots, d$-re) pedig $2^{d/2}$.

4.4. tétel. (*A mohó algoritmus lépésszáma.*) *A mohó algoritmus $\mathcal{B}_d$-n $O(2^{d/2})$ időben fut, a maximális sorhosszúság szintén $O(2^{d/2})$.*

4.2.2. Véletlenített algoritmus

A véletlenítés, mint oly sokszor korábban, itt is számottevően javíthatja az előbb említett mohó algoritmus tulajdonságait. Már a rácsokon értelmezett csomagirányítási feladatoknál is alkalmaztuk azt a megoldást, hogy a csomagot először egy véletlenszerűen választott közbülső állomásra küldjük, majd onnan továbbítjuk eredeti céljához. Ezzel a megoldással elérhetjük, hogy a csomagok nagy valószínűséggel nem találkoznak egyetlen másik csomaggal sem útjuk során, aminek következtében az egyes élek menti torlódás kialakulásának valószínűsége jelentősen csökken. Ezt a stratégiát követjük a pillangóhálózatok esetében is.

A probléma tehát az eredeti, a javasolt megoldás pedig a következő H´ -
´ ´ algoritmus:

1. fázis. A csomagot egy véletlenszerűen választott közbülső címre irányítjuk a d-edik szinten.

2. fázis. A csomagot az eredeti céljának megfelelő sorba irányítjuk, ám a nulladik szinten.

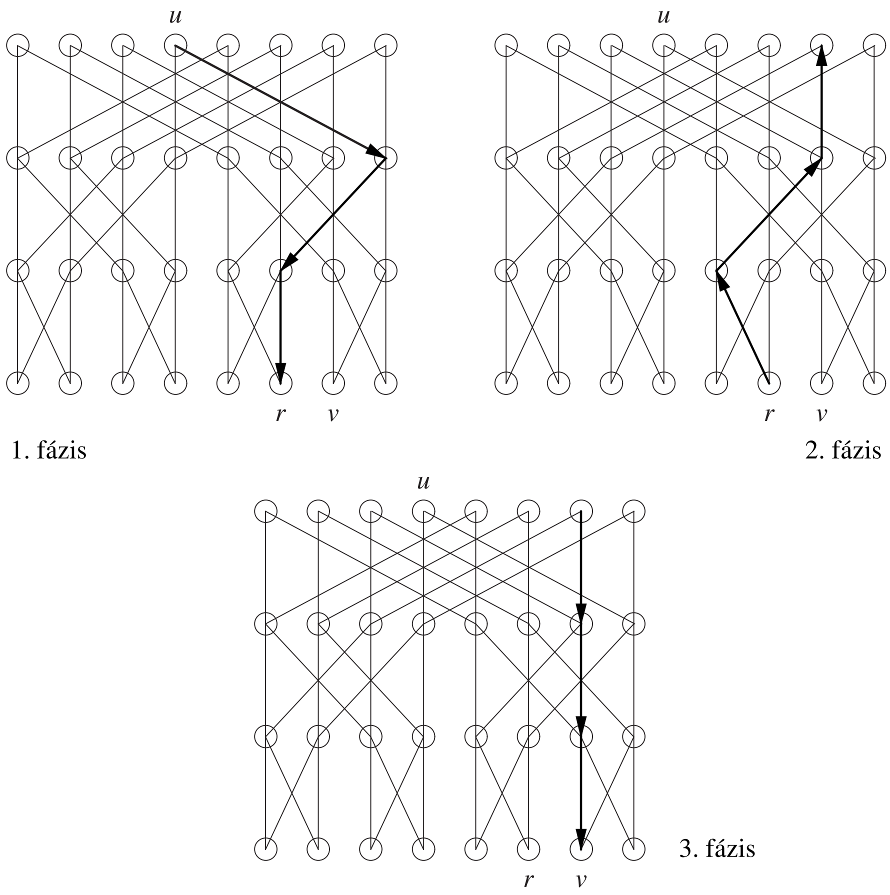
3. fázis. A csomagok elérik tényleges céljukat a d-edik szinten.

A 4.10. ábra szemlélteti az algoritmus 3 fázisát.

Az ábrán r a d-edik szint egy véletlenül választott csúcsa. u és v a vizsgált csomag kezdeti, illetve végső helye. A vastagított élek mutatják, hogy r az első fázisban eljut a $d$edik szintre, onnan a második fázisban a nulladik szintre, végül a harmadik fázisban a csomag célba ér.

A harmadik fázisban a csomagok végig a közvetlen éleken közlekednek, így akkor torlódás nem léphet fel, ezért a harmadik fázis végrehajtása pontosan d lépést igényel. A második fázis az első fázis *inverze*, tulajdonságai megegyeznek az első fázisával, így a továbbiakban elegendő azt vizsgálni, hogy megkapjuk a teljes algoritmus lépésszámát. Ehhez felhasználjuk majd a következő definíciókat és lemmát.

Legyen P utak egy halmaza egy hálózatban. Azt mondjuk, hogy P *nem ismét-*



4.10. ábra. Véletlenített csomagirányítás 3 fázisa

lődő úthalmaz (♣), ha bármely két útra a *P* halmazból igaz, hogy a metszetük üres vagy összefüggő, azaz ha találkoznak, akkor egy darabig együtt mennek, majd szétválás után nem találkoznak többé.

Két csomagot *átfedőnek* (♣) mondunk egy hálózatban, ha az általuk megtett utak legalább egy közös élt tartalmaznak.

4.5. lemma. (*Sorban állási lemma.*) *Legyen P utak halmaza egy hálózatban, ame-*

lyeken csomagok haladnak. Ha P nem ismétlődő úthalmaz, akkor tetszőleges p csomag késleltetése legfeljebb akkora, mint a p-vel átfedő csomagok száma.

Bizonyítás. Legyen p egy tetszőleges csomag. Ha egyetlen p-vel átfedő csomag sem késlelteti p-t egynél több alkalommal, akkor az állítás teljesül. Tegyük fel, hogy valamely q p-vel átfedő csomag kétszer is késlelteti p-t. Ekkor q maga is késleltetve volt egy p-vel átfedő másik csomag által, amely így már nem fogja késleltetni p-t. ■

4.2.3. Az első fázis elemzése

Legyen π egy tetszőleges csomag, jelölje továbbá e_i az az élt amelyen π az i-edik szinten áthalad ($1 \leq i \leq d$). A sorbaállási lemma alapján π késleltetésének meghatározásához elegendő meghatározni a π-vel átfedő csomagok számát. Jelöljük n_i-vel azon csomagok számát, amelyek útvonalában szerepel e_i. Ekkor

$$D = \sum_{i=1}^{d} n_i \tag{4.5}$$

felső korlát a π-vel átfedő csomagok számára. Tekintsük most az e_i élet. Ezen az élen legfeljebb $2^i - 1$ csomag halad keresztül, mivel ennyi processzor van a nulladik szinten amelyek mohó útvonalában szerepelhet e_i. Minden ilyen csomag $\frac{1}{2^i}$ valószínűséggel halad át az e_i élen mivel minden, a nulladik szintről induló, csomag $\frac{1}{2}$ valószínűséggel választja vagy a közvetlen- vagy a keresztélt minden szinten, egymástól függetlenül, i szinten keresztül, míg áthalad e_i-n. Így az n_i értéke $B(2^{i-1}, \frac{1}{2^i})$ binomiális eloszlású, melynek várható értéke $\frac{1}{2}$. Ebből következik, hogy D várható értéke a várható értékek összege, azaz $\frac{d}{2}$. Most megmutatjuk, hogy a teljes késleltetés nagy valószínűséggel $O(d)$. A D változónak $B(d, \frac{1}{2})$ egy felső korlátja. A Csernov-egyenlőtlenséget felhasználva az alábbi egyenlőtlenségeket kapjuk:

$$
\begin{aligned}
P[D > e\alpha d] &\leq \left(\frac{d/2}{e\alpha d}\right)^{e\alpha d} e^{e\alpha d - d/2} && (4.6)\\
&< \left(\frac{1}{2e\alpha}\right)^{e\alpha d} e^{e\alpha d} && (4.7)\\
&\leq \left(\frac{1}{2e\alpha}\right)^{e\alpha d} && (4.8)\\
&< 2^{-e\alpha d} && (4.9)\\
&< p^{-\alpha-1}, && (4.10)
\end{aligned}
$$

ahol $\alpha \geq 1$ és kihasználtuk azt a tényt, hogy $d = \Theta(\lg p)$. Mivel a csomagok száma p-nél kisebb, így annak a valószínűsége, hogy legalább egyikük $2e\alpha d$-nél nagyobb késleltetést szenved, kisebb, mint $p^{-\alpha-1}p = p^{-\alpha}$. Ezzel bebizonyítottuk a következő tételt.

4.6. tétel. (V´ - - ´ ´ *lépésszáma.*) *A* V´ - - - ´ ´ *algoritmus $\mathcal{B}_d$-n $\overline{O}(d)$ lépést tesz.*

Mivel bármely hálózatban a hálózat átmérője alsó korlát a csomagirányítási probléma maximális lépésszámára, így a fenti algoritmus nagy valószínűséggel aszimptotikusan optimális.

4.2.3.1. A sorméret elemzése

Az algoritmus várakozási sorok mérete szintén $\overline{O}(d)$. Legyen v_i egy i-edik szintű processzor. A v_i-n potenciálisan átmenő csomagok száma 2^i. Minden ilyen csomag $\frac{1}{2^i}$ valószínűséggel halad át v_i-n, így az áthaladó csomagok várható értéke 1. A Csernov-egyenlőtlenség felhasználásával megmutatható, hogy az áthaladó csomagok száma $\overline{O}(d)$.

4.3. Alapvető algoritmusok

Ebben a fejezetben alapvető problémák – üzenetszórás, prefixszámítás, adat koncentráció – megoldására mutatunk hiperkocka algoritmusokat. Mindezen algoritmusok lépésszáma $O(d)$. Mivel a hálózat átmérője minden nem triviális probléma megoldásánál alsó korlát a lépésszámra, így ezek aszimptotikusan optimális algoritmusok.

4.3.1. Üzenetszórás fában

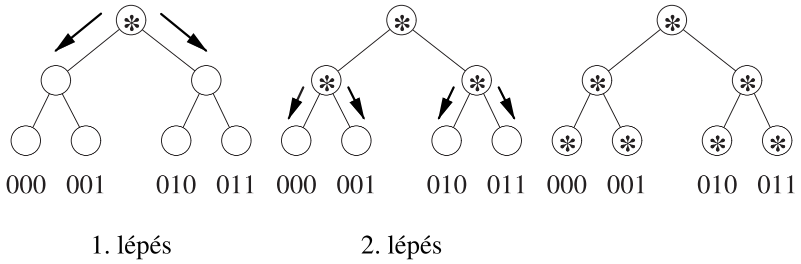
Az üzenetszórás problémája egy hálózatban azt a feladatot jelenti, amikor egy üzenetet valamely processzortól el kell juttatni a hálózathoz tartozó összes többi processzorhoz (vagy esetleg azok egy részhalmazához). Az üzenetszórást sokszor más algoritmusokba beépítve használjuk. A feladatot a F´ - ´ algoritmus segítségével oldjuk meg, amely a bináris fa architektúrának hiperkockába való beágyazásán alapul. Feltesszük, hogy az elküldendő üzenet a fa gyökerében található. Ez a processzor két másolatot készít a csomagról és elküldi a két leszármazottjának. Ha a fa egy belső processzora kap egy csomagot, akkor arról egy-egy másolatot továbbküld gyerekeinek. Ez folytatódik mindaddig, amíg minden levélen megjelenik az üzenet egy példánya.

4.7. tétel. (*Üzenetszórás fában.*) *A fában való üzenetszórást a* F´ - ´ *algoritmus egy soros* $\mathcal{H}_d$ *hiperkockán* $\Theta(d)$ *lépésben oldja meg.*

Bizonyítás. A beágyazott bináris fa magassága d, így $O(d)$ lépés után minden levél processzor ismerni fogja az üzenetet. Az algoritmus végrehajtásakor minden időegységben a fának pontosan egy szintjén elhelyezkedő processzorok dolgoznak, más szóval az algoritmus normális, így egy lépése – amint azt a bináris fáknak a hiperkockába ágyazásakor megmutattuk – $\mathcal{H}_d$-nek pontosan egy lépésével szimulálható.

Másrészt legrosszabb esetben legalább egy levélhez el kell juttatni az üzenetet, ezért a fenti beágyazásra $W(d, \text{F´ -} \quad ´ \quad) = \Omega(d)$. ■

4.8. példa. Üzenetszórás 4 levelű fában A 4.11. ábra egy 3 szintes fát mutat, melyben a gyökérben lévő üzenetet juttatjuk el a többi csúcshoz. Az ábra első része azt mutatja, hogy az első lépésben az üzeneteket a gyökércsúcs gyerekei kapják meg. A középső részen az üzenetek a gyerekek gyerekeihez, azaz a levélcsúcsokhoz jutnak el. Végül az ábra harmadik része a végső állapotot mutatja, amelyben már minden csúcs megkapta az üzenetet. Mivel a fát úgy ágyaztuk be a hiperkockába, hogy a fa különböző szintjein lévő processzorok a hiperkocka különböző processzorára képződjenek le, $\mathcal{H}_2$ két lépésben megoldja a feladatot. ♠



4.11. ábra. Üzenetszórás 4 levelű fában

4.3.2. Prefixszámítás fán

Ezen feladat megoldásához ismét a bináris fa beágyazását fogjuk felhasználni. Legyen x_i a bemenet a 2^d levelű bináris fa i-edik levelén. Az alább bemutatott P - ´ algoritmus két fázisban számítja ki az eredményt. Az első (felfelé haladó) fázisban az adatok felfelé áramlanak a bináris fában, a gyökér felé. A második fázisban (lefelé haladó) az adatok a gyökér felől haladnak a levelekhez. Mindkét fázis minden lépésében a bináris fának pontosan egy szintje vesz részt a számításban, azaz az algoritmus normális.

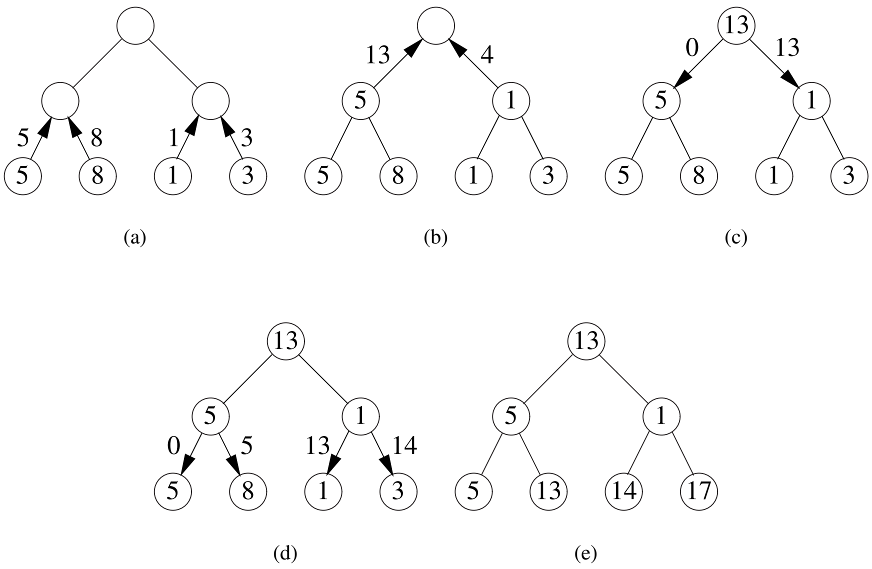
Felfelé haladó fázis. A levelek elküldik az x_i-ket a szülő csúcsoknak. A fa egy tetszőleges belső csúcsa, amennyiben két elemet kap (y-t a bal és z-t a jobboldali leszármazottjától), akkor kiszámítja a $w = y + z$ értéket, tárolja w-t és y-t, majd elküldi w-t a szülőjének. Az első fázis végére minden processzor kiszámítja és tárolja a hozzá tartozó részfában levő bemeneti elemek összegét. Így a gyökér az összes elem összegét tartalmazza.

Lefelé haladó fázis. A gyökérben levő processzor elküld egy nullát a bal oldali gyerekének és y-t a jobboldalinak. Egy tetszőleges belső processzor ha q értéket kapja a szülőjétől, akkor elküldi q-t a baloldali és $q \oplus y$ értéket a jobboldali gyereké-nek. Az i-edik levél processzor, ha a szülőjétől a q értéket kapja, akkor kiszámolja és végeredményként tárolja az $x_i + q$ értéket.

Az algoritmus felfelé haladó fázisában minden processzor a hozzá tartozó részfa leveleiben tárolt számok összegét számolja ki. Legyen P_v egy processzor és jelöljük $P_{v'}$-vel P_v részfájának legbaloldalibb levelét. Ekkor a lefelé haladó fázis során a P_v által kapott q értéke, azaz q a $P_{v'}$-től balra eső elemek összege. Az észrevétel felhasználásával az algoritmus helyessége egyszerűen bizonyítható. Az algoritmus mindkét fázisa $d - 1$ lépést igényel. Mivel minden időpillanatban a fának pontosan egy szintje aktív, így az algoritmus minden lépése szimulálható $\mathcal{H}_d$ egy lépésével.

4.9. példa. Prefixszámítás fán

A 4.12. ábrán egy 4-levelű fa leveleiben vannak az 5, 8, 1 és 4 számok. A feladat a megfelelő prefixek számolása, ha az elvégzendő művelet az összeadás. Az ábra (a) része szerint minden levél elküldi a tárolt számot a szülőjének. A (b) ábrarész szerint a szülők tárolták a baloldali gyereküktől kapott értéket (a kiszámított összeget is, de ezt az ábra nem mutatja), és szülőjüknek elküldték a kapott 2 szám összegét. ♠



4.12. ábra. Prefixszámítás bináris fán

4.10. tétel. (*Prefixszámítás fán és pillangón.*) *A* P - ´ *algoritmus a prefixszámítást* 2^d *levelű bináris fán, valamint* $\mathcal{H}_d$*-n is* $\Theta(d)$ *lépésben oldja meg.*

Összegzési feladatnak (♣) nevezzük az $x_1 \oplus x_2 \oplus \cdots \oplus x_n$ kifejezés értékének kiszámítását adott x_i-khez. A fenti algoritmus felfelé haladó fázisának végére kiszámítja ezt az összeget, így az összegzési feladat megoldásához szükséges idő a fele az összes prefix kiszámításához szükséges időnek.

4.3.3. Adatkoncentráció

Legyen egy $\mathcal{H}_d$ hiperkocka $k(k < p)$ processzorán egy-egy elem elhelyezve. A feladat, hogy mozgassuk ezen elemeket a hiperkocka $P_0, P_1, \ldots, P_{k-1}$ processzoraira (processzoronként egy elemet elhelyezve).

Ha meg tudjuk határozni, hogy melyik elemet melyik processzorra kell juttatnunk, akkor a 4.2.2. szakaszban vizsgált véletlenített H´ - ´ ´ csomagirányítási algoritmus segítségével $O(d)$ lépésben megtehetjük ezt. Ez az algoritmus párhuzamos (többportos) hiperkockát feltételez.

Van azonban egy jóval egyszerűbb, determinisztikus algoritmus is a feladat megoldására. Pontosabban a feladatra adunk egy normális megoldást pillangóhálózaton, majd felhasználjuk az ide vonatkozó beágyazási lemmát. Előtte azonban vizsgáljuk meg a pillangóhálózatok két tulajdonságát, amelyekre az algoritmus elemzésekor hivatkozni fogunk.

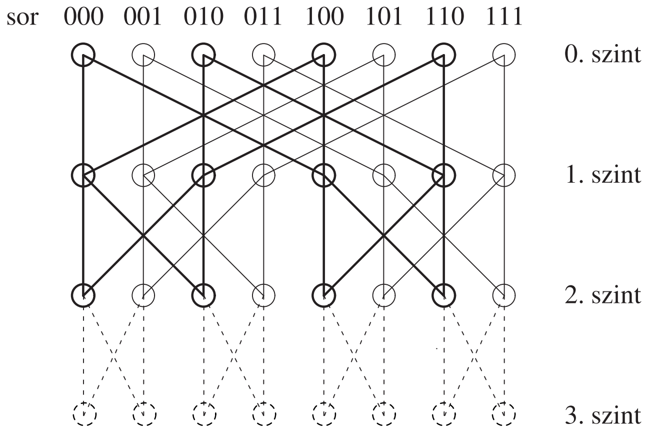
1. tulajdonság. Ha egy pillangóhálózatból eltávolítjuk a d-edik szint összes processzorát és a hozzájuk kapcsolódó éleket, két $d-1$ szintes pillangóhálózatot kapunk eredményül. Az egyik hálózat az eredetinek a páros sorait tartalmazza, így ezt *páros alhálózatnak* nevezzük (♣), a másik a páratlanokat, így az a *páratlan alhálózat* (♣).

2. tulajdonság. A nulladik szint bármely processzora a leszármazottaival együtt egy 2^d levelű bináris fát alkot, amelynek levelei pontosan a d-edik szintű processzorok.

Ezek után felvázolhatjuk az adatkoncentráció problémájának megoldó algoritmusát.

4.3.4. Kisszámú elem rendezése hiperkockán

A *ritka leszámláló rendezés* problémája már szerepelt a harmadik fejezetben. Legyen $X = k_0,\ k_1,\ \ldots,\ k_{\sqrt{p}-1}$, ahol $p = 2^d$. Az általánosság megszorítása nélkül



4.13. ábra. A d-edik szinten lévő processzorok és élek eltávolítása

feltehető, hogy d páros. Tegyük fel, hogy a bemenő adatok a hiperkocka azon processzorain vannak (processzoronként egy kulcs), melyek címkéjét úgy kapjuk, hogy a címkék d bitje közül az első $\frac{d}{2}$-t nullának választjuk. A rendezett sorozatot ugyanezen processzorokon állítjuk elő kimenetként.

Legyen P_v a $\mathcal{H}_d$ hiperkocka egy processzora. P_v címkéje $\langle i, j\rangle$ párként is felfogható, ahol i az első $\frac{d}{2}$ bit, j pedig a második $\frac{j}{2}$ bit. Azok a processzorok, melyek címkéjének első fele i $(0 \leq i \leq 2^{d/2} - 1)$, egy $\mathcal{H}_d$ hiperkockát alkotnak. Ezt a hiperkockát $\mathcal{H}_d$ *i-edik sorának* (♣) fogjuk nevezni. Hasonlóképpen határozzuk meg a hiperkocka *j-edik oszlopát* (♣). A rendezendő kulcsok kezdetben a nulladik sor processzorain vannak. Az egyértelműség kedvéért tegyük fel, hogy az r rangú kulcs kimenetként a $\langle 0, r - 1\rangle$ $(0 \leq r \leq 2^{d/2})$ processzoron fog megjelenni.

A K - ´ algoritmus a következőképpen működik.

K - ´ (X, Y) *párhuzamos eljárás*

Számítási modell: hiperkocka

Bemenet: $X = k_0,\ k_1,\ \ldots, k_{\sqrt{p}-1}$ (rendezendő kulcsok)

Kimenet: $Y = l_0,\ l_1,\ \ldots, l_{\sqrt{p}-1}$ (rendezett kulcsok)

O_j **in parallel for** $0 \leq j \leq \sqrt{p} - 1$
    Szórja k_j-t úgy, hogy minden sor
    megkapja X egy másolatát
S_i **in parallel for** $0 \leq i \leq \sqrt{p} - 1$
    Kiszámítja k_i rangját úgy, hogy a sor
    minden processzorához szétszórja k_i-t,
    majd a bemenet minden elemével
    összehasonlítja, és összeget számol
S_i **in parallel for** $0 \leq i \leq \sqrt{p} - 1$
    szétszórja a sorban k_i rangját
S_i **in parallel for** $0 \leq i \leq \sqrt{p} - 1$
    if $r(k_i) = r_i$ **then**
        $\langle i, r_i - 1 \rangle$ szórja k_i-t O_i
        processzoraihoz úgy, hogy
        végül $\langle 0, r_i - 1 \rangle$ megkapja
        azt a kulcsot, amelynek
        kiviteléért felelős

A lépészám a következő.

4.11. tétel. (*Ritka leszámláló rendezés kockán.*) *A* K - - ´ *algoritmus legfeljebb* $\sqrt{p}$ *kulcs ritka leszámláló rendezését egy* $\mathcal{H}_d$ *hiperkockán* $\Theta(d)$ *lépésben végzi el.*

Bizonyítás. Az algoritmus csak olyan műveleteket végez, amelyben egy sor vagy oszlop processzorai vesznek részt. Az elvégzett műveletek mindegyike – prefixszámítás, üzenetszórás, összehasonlítás – $O(d)$ lépésben elvégezhető, és legrosszabb esetben $\Omega(d)$ lépést igényel. ■

4.4. Kiválasztás

Az egyszerű kiválasztás célja, hogy adott n kulcs közül az i-edik ($1 \leq i \leq n$) legkisebbet meghatározzuk. A korábbiakhoz hasonlóan két esetet vizsgálunk: az egyikben a processzorok p száma megegyezik a kulcsok n számával, a másikban pedig a processzorok száma kisebb, mint a kulcsoké.

Három algoritmust mutatunk be, mindegyik pillangó hálózaton fut. Az elsőnél $p = n$, a másik kettőnél $p < n$. Az első nagy valószínűséggel munkahatékony, a másik kettő viszont nem az.

4.4.1. Véletlenített algoritmus a $p = n$ esetre ($*$)

A PRAM-on futó munkahatékony véletlenített algoritmus alkalmazható hiperkockán is.

A K ´ - ´ fokozatainak száma $\overline{O}(1)$. A PRAM- - ´ első lépése hiperkockán $O(1)$ lépést vesz igénybe. A 2. és 5. lépésekben szükséges prefixszámítás összesen $O(d)$ lépésben elvégezhető. A 3. és 6. lépésekben a koncentrálás ugyancsak megoldható $O(d)$ lépésben. A 3. és 6. lépésekben szükséges ritka rendezés ugyanennyi lépést vesz igénybe. Mivel a 4. és 6. lépésben rendezett sorozatból kell kiválasztani, ezért az $O(1)$ lépésben megoldható. A 2., 4. és 5. lépésben lévő üzenetszórás lépésigénye $O(d)$. Innen adódik a következő tétel.

4.12. tétel. (*Kiválasztás hiperkockán.*) *Ha $p = n$, akkor a* K ´ - ´ *algoritmus a kiválasztási feladatot a $\mathcal{H}_d$ hiperkockán $\overline{O}(d)$ idő alatt megoldja.*

4.4.2. Véletlenített algoritmus a $p < n$ esetre ($*$)

Tegyük fel, hogy $\epsilon > 1$ és $n = p^{\epsilon}$. A PRAM modellen futó V´ - ´ algoritmus módosított változata a rácsokhoz hasonlóan most is alkalmazható.

A K ´ - ´ - ´ algoritmus esetén minden processzor $\frac{n}{p}$ kulccsal kezd. A **while** utasítás feltételét $(N > D)$-re változtatjuk (ahol D állandó). Az első lépésben minden processzor minden kulcsa $\frac{1}{N^{1-(1/3c)}}$ valószínűséggel lesz a mintában.Ezért ez a lépés ebben az esetben $\frac{n}{p}$ lépésig tart. A második lépés változatlan és $O(d)$ lépést vesz igénybe. A 3. lépésben a koncentrálás és a ritka leszámláló rendezés is végrehajtható $O(d)$ lépésben. A 4., 5. és 6. lépések szintén végrehajthatók $O(d)$ lépésben. Így minden fokozat $O(\frac{n}{p} + d)$ lépést vesz igénybe. Az algoritmusnak $\overline{O}(\lg\lg n)$ fokozatra van szüksége. A 6 lépésre külön kapott becslések összegét a fokozatszámra kapott becsléssel összeszorozva kapjuk a következő tételt.

4.13. tétel. (*Kiválasztás hiperkockán.*) *Ha $c > 1$ és $n = p^c$, akkor a kiválasztási feladat a $\mathcal{H}_d$ hiperkockán $\overline{O}\left((\frac{n}{d} + d)\lg\lg p\right)$ idő alatt megoldható.*

4.4.3. Determinisztikus algoritmus a $p < n$ esetre

A négyzeten futó kiválasztó algoritmus adaptálható hiperkockára. Az így kapott K ´ - - ´ algoritmus lépésszámának elemzése hosszú, csak az eredményét adjuk meg.

4.14. tétel. (*Kiválasztás hiperkockán.*) *Ha $p < n$, akkor a* K ´ - - ´ *algoritmus a kiválasztási feladat a $\mathcal{H}_d$ hiperkockán $O\left(\frac{n}{p}\lg\lg p + d^2\lg n\right)$ lépésben megoldja.*

4.15. példa. Legnagyobb elem kiválasztása hiperkockán

Tegyük fel, hogy a $\mathcal{H}_3$ hiperkocka minden processzorának memóriájában 5-5 kulcsot helyezünk el – ahogyan azt a 4.14. ábra mutatja. A feladat a harminckettedik legkisebb elem kiválasztása.

Először minden processzor meghatározza saját kulcsainak mediánját – ezeket az ábra felső részén bekarikáztuk. Ezen mediánok rendezett sorozata 6, 16, 18, 22, 25, 26, 45, 55. Mivel most minden processzornál 5-5 kulcs van, a súlyozott médián megegyezik a mediánnal, ami 22. Ezután meghatározzuk $M = 22$ rangját, ami 21. Mivel $i = 32 > 21$, a 22-nél kisebb vagy vele egyenlő kulcsokat elhagyjuk. i frissített értéke $32 - 21 = 11$ lesz. Ezzel a **while** ciklus egy menetét befejeztük.

A **while** ciklus következő menetében 19 elemmel kezdünk. Az ábra középső részén bekarikáztuk a helyi mediánokat. A mediánok rendezett sorozata 23, 24, 27, 28, 35, 36, 45, 63, a megfelelő súlysorozat pedig 1, 2, 1, 3, 2, 3, 4, 3, így a súlyozott médián 36. Elhagyjuk a kis kulcsokat és $i = 11 - 10 = 1$ lesz az új i érték. Ezzel vége a **while** ciklus következő menetének.

Így folytatva megkapjuk, hogy a kiválasztás eredménye a megmaradt 9 szám közül a legkisebb, a 42. ♠

Processzor	000	001	010	011	100	101	110	111
	5	20	18	35	63	21	62	11
	10	15	24	42	71	9	51	28
	6	7	12	3	1	36	45	17
	4	16	13	19	2	47	8	81
	27	23	32	22	55	26	30	25

A súlyozott médián 22.

	27	23	24	35	63	36	62	28
			32	42	71	47	51	81
					55	26	45	25
							30	

A súlyozott médián 36.

	-	-	-	42	63	47	62	81
					71		51	
					55		45	

4.14. ábra. Kiválasztás 8 elem közül

4.5. Összefésülés

Amint már láttuk az előző két fejezetben, az összefésülés célja, hogy két rendezett sorozatból a két sorozat minden elemét tartalmazó, rendezett sorozatot állítsunk elő.

Beláttuk, hogy ez a feladat hiperkockán – m hosszúságú bemenő sorozatokra – m^2 processzoron $O(\lg m)$ idő alatt megoldható. A számítási modell most is a hiperkocka lesz, de csak $2m$ processzort használunk (feltéve, hogy m 2 hatványa).

4.5.1. Páratlan-páros összefésülés

Legyen $X_1 = k_0,\ k_1,\ \ldots, k_{m-1}$ a két összefésülendő rendezett sorozat, ahol $2m = 2^d$.

Először szétválasztjuk X_1 és X_2 páros és páratlan részét. Legyenek ezek $O_1,\ E_1,\ O_2$ és E_2. Ekkor E_1-et rekurzívan összefésüljük O_2-vel, az eredmény $A = a_0, a_1, \ldots, a_{m-1}$. Ugyanígy kapjuk $B = b_0, b_1, \ldots, b_{m-1}$-et O_1-ből és E_2-ből. Ezután A és B összekeverésével kapjuk a $C = a_0, b_0, a_1, b_1, \ldots, a_{m-1}, b_{m-1}$ sorozatot, majd rendre összehasonlítjuk (és szükség esetén felcseréljük a_i-t és b_i-t $(0 \leq i \leq m-1)$. Az algoritmus helyessége belátható a 0-1-elv segítségével.

4.16. példa. 2×4 **elem összefésülése**

Legyen $X_1 = 7, 11, 24, 30$ és $X_2 = 2, 4, 27, 45$. Ebben az esetben $O_1 = 11, 30$ és $E_1 = 7, 24$, $O_2 = 4, 45$ és $E_2 = 2, 27$. E_1 és O_2 összefésülésével $A = 4, 7, 24, 45$ adódik. Hasonlóképpen $B = 2, 11, 27, 30$. A és B összekeverésének eredménye $C = 4, 2, 7, 11, 24, 27, 45, 30$. Ha a 4-et és 2-t, valamint a 45-öt és 30-at felcseréljük, akkor az eredmény 2, 4, 7, 24, 27, 30, 45. ♠

A módosított algoritmust könnyű a $\mathcal{B}_d$ pillangón megvalósítani. Például X_1 és X_2 páros és páratlan részre való felbontása a pillangó hálózaton egy lépésben elvégezhető.

A keverési művelet szintén könnyen elvégezhető. Tegyük fel, hogy X_1 és X_2 is a $\mathcal{B}_d$ pillangó hálózat d-edik szintjének bemenő adatai. Legyen X_1 a bemenet az első m sorban és X_2 a második m sorban. Az algoritmus első lépése X_1 és X_2 szétválasztása páratlan és páros részre. Ezután rekurzívan összefésüljük E_1-et O_2-vel és O_1-et E_2-vel. Ennek érdekében az első m sorban lévő kulcsokat a közvetlen éleken, a többi kulcsot a keresztéleken mozgatjuk (lásd 4.15. ábra).

4.17. tétel. *Ha $m = 2^d$, akkor két m hosszúságú lista mind a $\mathcal{B}_d$ pillangó hálózaton, mind pedig a $\mathcal{H}_d$ hiperkocka hálózaton összefésülhető $O(d)$ idő alatt.*

Processzor	000	001	010	011	100	101	110	111
	5	20	18	35	63	21	62	11
	10	15	24	42	71	9	51	28
	6	7	12	3	1	36	45	17
	4	16	13	19	2	47	8	81
	27	23	32	22	55	26	30	25

A súlyozott médián 22.

27	23	24	35	63	36	62	28
		32	42	71	47	51	81
				55	26	45	25
						30	

A súlyozott médián 36.

-	-	-	42	63	47	62	81
				71		51	
				55		45	

4.15. ábra. Páratlan-páros összefésülés pillangón

4.5.2. Biton összefésülés

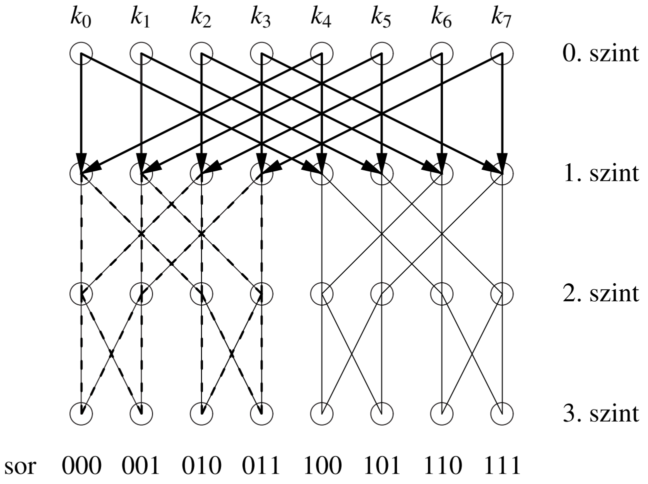
A $K = k_0,\ k_1,\ \ldots,\ k_{n-1}$ sorozatot *biton sorozatnak* (♣) nevezzük, ha rendelkezik a következő két tulajdonság valamelyikével:

a) van olyan j $(0 \leq j \leq n-1)$ index, amelyre $k_0 \leq k_1 \leq \ldots \leq k_j$ és $k_j \geq k_{j+1} \geq \ldots \geq k_{n-1}$;

b) van olyan j $(1 \leq j \leq n-1)$ index, amelyre a K sorozat $K' = k_{j+1},\ k_{j+2},\ \ldots,\ k_{n-1},\ k_1, k_2,\ \ldots,\ k_{j-1}$ ciklikus eltoltja rendelkezik az előző tulajdonsággal.

Megmutatjuk, hogyan lehet rendezni egy biton sorozatot pillangó hálózaton.

Az első lépésben – ahogyan ezt a 4.16. ábra mutatja – a nulladik szint processzorai mind a közvetlen, mind a kereszt éleken elküldik a kulcsukat. Az első szint processzorai felülről 2-2 kulcsot kapnak.



4.16. ábra. Bitonikus összefésülés pillangó hálózaton

4.18. tétel. *Egy* 2^d *hosszúságú biton sorozat mind a* $\mathcal{B}_d$ *pillangó hálózaton, mind pedig a* $\mathcal{H}_d$ *hiperkocka hálózaton rendezhető* $O(d)$ *idő alatt.*

4.6. Rendezés

Ebben az alfejezetben hiperkocka és pillangó hálózat lesz a számítási modell.

4.6.1. Páratlan-páros összefésülő rendezés

Első algoritmusunk a P´ - ´ - ´ ¨ - egy megvalósítása.

Jelöljük $R(d)$-vel algoritmusunk lépésszámát $\mathcal{B}_d$-n. Ekkor

$$R(d) = R(d-1) + O(d). \tag{4.11}$$

Ennek a rekurzív egyenletnek a megoldása $S(d) = O(d^2)$.

4.19. tétel. *Ha $p = 2^d$, akkor p elem mind a $\mathcal{B}_d$ pillangó hálózaton, mind pedig a $\mathcal{H}_d$ soros hiperkockán rendezhető $O(d^2)$ idő alatt.*

4.6.2. Biton rendezés

Az összefésülő rendezés alapötlete a biton összefésülő algoritmussal kapcsolatban is alkalmazható – az eredmény a P ´ - algoritmus.

Most ismét felírhatjuk a 4.11 rekurzív egyenletet, ahonnan a következő lépésszámot kapjuk.

4.20. tétel. (*Biton sorozat rendezése pillangón és hiperkockán.*) *A P ´ - - algoritmus $p = 2^d$ elemet $O(d^2)$ lépésben rendez a $\mathcal{B}_d$ pillangó hálózaton, és ugyancsak $O(d^2)$ lépésben hiperkockán.*

4.7. Gráfalgoritmusok

Ebben az alfejezetben minmátrix, tranzitív lezárt, összefüggő komponensek és minimális feszítőfa meghatározására szolgáló algoritmusokat mutatunk be.

4.7.1. Minmátrix meghatározása

Újra alkalmazzuk a második fejezetben bevezetett formalizmust.

Legyen q egy pozitív egész szám, $n = 2^q$ és legyen $M[0 : n-1, 0 : n-1]$ egy $n \times n$ méretű mátrix. Az M mátrix minmátrixának számítására a $\mathcal{H}_{3q}$ hiperkockát fogjuk használni, amelyben 2^{3q} processzor van, melyek címkéi $3q$ bit hosszúak. Ezek a címkék $\langle i, j, k\rangle$ alakban is felírhatók, ahol i a címke első q bitjét, j a címke második q bitjét és k a címke harmadik q bitjét jelentik.

Jelölje $(i, \star, \star)$ $(0 \le i \le 2^q - 1)$ (♣) a $\mathcal{H}_{3q}$ hiperkocka azon processzorait, melyek címkéjében az első q bit bináris számként i. Ezek a processzorok együtt egy $\mathcal{H}_{2q}$ hiperkockát alkotnak. Hasonlóképpen definiáljuk a $(\star, j, \star)$, $(\star, \star, k)$, $(i, j, \star)$, $(i, \star, k)$ és $(\star, j, k)$ jelöléseket is.

Ekkor a K ´ - ´ algoritmus a következőképpen működik.

K ´ - (M,$\widetilde{M}$) *párhuzamos eljárás*

Számítási modell: hiperkocka

Bemenet: M (egész elemeket tartalmazó mátrix)

Kimenet: M minmátrixa

A $q[i, j, k]$ elemek frissítése
Az $m[i, j]$ értékek frissítése

A lépészám a következő.

4.21. tétel. *Ha M egy $n \times n$ méretű mátrix, akkor a* K ´ - - *algoritmus M minmátrixát egy $\mathcal{H}_{3l}$ hiperkockán $O(\lg^2 n)$ lépésben meghatározza.*

Bizonyítás. Az algoritmus első szakaszában az üzenetszórás és a mátrixtranszponálás $O(\lg n)$ lépésben végrehajtható. A második szakaszban az $m[i, j]$ értékek frissítése prefixszámítással megoldható $O(lgn)$ lépésben. A négyzeten futó algoritmusban szereplő két üzenetszórás helyett itt a hiperkockában üzenetszórást végző algoritmust alkalmazzuk, ugyancsak $O(\lg n)$ lépésszámmal.

Tehát a **for** ciklus magjának minden végrehajtása $O(\lg n)$ lépést vesz igénybe. Mivel a ciklusmagot $\lg n$-szer kell végrehajtani, ebből a lépésszám felső korlátja már adódik.

Mivel például a második szakaszban végzett prefixszámítás lépésszáma $\Omega(n)$, ezért az elemzett algoritmus lépészámának nagyságrendje valóban $\lg^2 n$. ∎

4.7.2. Tranzitív lezárt

Irányított gráf tranzitív lezártját a második fejezetben definiáltuk.

4.22. tétel. *Egy n-csúcsú irányított gráf tranzitív lezártja $O(\lg^2 n)$ lépésben meghatározható egy n^3 processzort tartalmazó hiperkockán.*

Bizonyítás. Állításunk a 4.21. tétel következménye. ∎

4.7.3. Összefüggő komponensek

Gráfok összefüggő komponenseit a második fejezetben definiáltuk.

4.23. tétel. *Egy n csúcsú irányított gráf összefüggő komponensei egy n^3 processzort tartalmazó hiperkockán $O(\lg^2 n)$ lépésben meghatározhatók.*

Bizonyítás. Állításunk a 4.21. tétel következménye. ∎

4.7.4. Legrövidebb utak

A H ´ - algoritmus lépésszámára vonatkozik a következő tétel.

4.24. tétel. *A H ´ - algoritmus egy n-csúcsú irányított gráf összes csúcspárjának távolságát meghatározza $O(\lg^2 n)$ lépésben egy $\frac{n^3}{\lg n}$ processzort tartalmazó hiperkockán.*

Bizonyítás. A PRAM modellek elemzése során megmutattuk, hogy egy gráf csúcsai közötti távolságokat $\lg n$ mátrixszorzás segítségével meghatározhatók.

Két $n \times n$ méretű mátrix egy $\frac{n^3}{\lg n}$-processzoros hiperkockán $O(\lg n)$ lépésben összeszorozható. ∎

4.7.5. Konvex burok

A konvex burok számítására – hasonlóan a PRAM és négyzet modellek esetéhez – *oszd meg és uralkodj* elvű algoritmust javaslunk.

Ha a rekurzív algoritmusunk lépésszámát a d-dimenziós hiperkockán $K(d)$-vel jelöljük, akkor

$$K(d) = K(d-1) + O(d^2), \tag{4.12}$$

ahonnan $K(d) = O(l^3)$.

4.25. tétel. *Ha $n = 2^d$, akkor egy sík n pontjának konvex burka $O(\lg^3 n)$ lépésben meghatározható a $\mathcal{H}_d$ hiperkockán.*

4.8. Gyakorlatok

4-1. Állapítsuk meg, van-e Euler-kör és/vagy Hamilton-kör a vizsgált hálózatokban?

4-2. Mennyi idő alatt lehet egy üzenetet eljuttatni a fejezetben vizsgált hálózatok adott processzorától az összes többihez? Tervezzünk bejárási algoritmusokat és elemezzük lépésszámukat.

4-3. Mutassuk meg, hogy egy pillangó hálózat első szintjén lévő processzorból pontosan egy út vezet a d-edik szint bármelyik processzorához.

4-4. Foglaljuk táblázatba a vizsgált hálózatok jellemző tulajdonságait.

4-5. Hányféleképpen lehet beágyazni a 4.3. ábra bal oldalán látható G hálózatot az ábra jobb oldalán látható H hálózatba?

4-6. Adjuk meg a $\mathcal{G}_3$, $\mathcal{G}_4$ és $\mathcal{G}_5$ Gray-kódokat.

4-7. Tervezzünk $O(kd)$ lépésszámú algoritmust, amely 2^d k-bites kulcsnak a $\mathcal{B}_d$ pillangó hálózaton való rendezésére.

4.9. Feladatok

4-1. Gray-kód elemzése

Adjuk meg, összesen hány nullát és hány egyest tartalmaz $\mathcal{G}_k$ ($k \geq 1$). Hány olyan elrendezése van a k-jegyű bináris számoknak, melyekre jellemző, hogy az i-edik ($2 \leq i \leq 2^k$) szám pontosan egy helyen tér el az $(i-1)$-ediktől? Hány ilyen ciklikus elrendezés van?

4-2. Rács beágyazása hiperkockába

Tekintsük egy 4×8 méretű rácsnak $\mathcal{H}_5$-be ágyazását. Az első két bitet használjuk a rács sorainak, a további három bitet pedig a rács oszlopainak azonosítására: $\mathcal{G}_2$ feleljen meg a nulladik, első, második, illetve harmadik sornak. Ugyanígy $\mathcal{G}_3$ elemei rendre megfelelnek a $0.,\ 1.,\ \ldots, (n-1)$. oszlopnak. Adjuk meg a beágyazás jellemző adatait (felfúvódás, késleltetés, torlódás).

4-3. Teljes bináris fa beágyazása de Bruijn-hálózatba

Mutassuk meg, hogy egy d-szintes teljes bináris fa beágyazható egy d-dimenziós de Bruijn-hálózatba úgy, hogy a beágyazás késleltetése 1.

4-4. Mohó algoritmus korlátjának élessége

Lássuk be, hogy a mohó algoritmus lépésszámára és sorhosszúságára bizonyított felső korlát aszimptotikusan éles, azaz van csomagirányítási feladatoknak olyan sorozata amelyre a (4.4) egyenletben megadott nagyságrend a jellemző. *Útmutatás.* Vizsgáljuk meg azt az úgynevezett bitfordításos feladatot, ahol a $b_1...b_d$ sorbeli csomag címzettje a $b_d...b_1$ sorbeli processzor. Vizsgáljuk meg ebben az esetben egy $\frac{d}{2}$-edik szintű él forgalmát.

4-5. Polinomok szorzása

Tervezzünk algoritmust, amely két 2^d-edfokú polinomot $O(d)$ idő alatt összeszoroz a $\mathcal{B}_d$ pillangó hálózaton.

4-6. Gyors Fourier transzformáció

Tervezzünk algoritmust, amely a $\mathcal{B}_d$ pillangó hálózaton kiszámítja egy 2^d hosszúságú vektor gyors Fourier-transzformáltját.

5. Szinkronizált hálózat

Ebben a fejezetben döntési feladatokat oldunk meg – speciális és általános hálózatokban.

5.1. Számítási modell

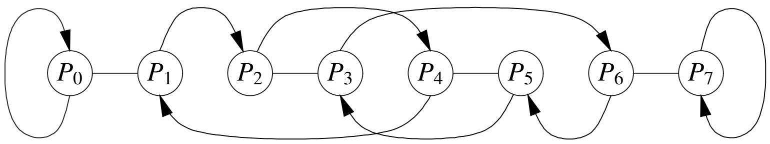
A H hálózatot a korábbiakhoz hasonlóan $H = (V, E)$ formában adjuk meg, ahol $V = \{P_1,\ P_2,\ \ldots,\ P_p\}$ a processzorok halmaza. Két processzor között

- vagy kétirányú adatátvitel lehetséges,
- vagy csak egyirányú adatátvitel van,
- vagy nincs kapcsolat.

Ennek megfelelően az élek E halmaza két részből áll: $E = (S, D)$, ahol S az egyirányú adatátviteli vonalakat leíró irányított élek halmaza, míg D a kétirányú adatátvitelt leíró irányítatlan élek halmaza.

A keverő-cserélő hálózatokban kétféle él van: kétirányú *cserélő* (♣) és egyirányú *keverő* (♣) él.

A d dimenziós *teljes keverő-cserélő* hálózatban (♣) $p = 2^d$ processzor van. A cserélő élek a P_{2i} processzorból a P_{2i+1} ($i = 0,\ 1,\ \ldots, 2^{d-1} - 1$) processzorhoz vezetnek. Minden processzorból egy keverő él indul: a P_i ($i = 0,\ 1,\ \ldots\ 2^p - 1$) processzorból induló keverő él a P_{2i} processzornál végződik, ahol az indexeket (mod $2^p - 1$) vesszük.



5.1. ábra. 8-processzoros keverő-cserélő hálózat

A 5.1. ábra egy 8-processzoros teljes keverő-cserélő hálózatot ábrázol.

A gyakorlatban használt hálózatok többségében csak kétirányú adatátviteli vonalak vannak. A de Bruijn-hálózat csak egyirányú adatátvitelt enged meg.

A P_i processzor *szomszédait* (♣) *szomszéd*[i]-vel jelöljük és a következőképpen definiáljuk:

$$szomszéd[i] = \{P_j \mid (i, j) \in S \lor (i, j) \in D \lor (i, j) \in D)\}. \tag{5.1}$$

A processzorokat automataként írjuk le, amelyek a szinkronizált lépésekben üzeneteket küldhetnek és kaphatnak, és adott kezdőállapotból kiindulva minden lépésben – a beérkező üzenetek és a korábbi állapot által meghatározott – új állapotba mennek át.

A P_i processzor a K¨ $_i$(*üzenet*) és a F $_i$(*üzenet*) függvénnyel küldenek, illetve fogadnak üzenetet. Az *üzenet* az *Ü*ϵ halmaz eleme, ahol *Ü* a lehetséges üzenetek halmaza, ϵ pedig az üres üzenet. Egy üzenet lehet például a küldő folyamat azonosítója, és állhat több részből is.

Ebben a fejezetben a lépésszám mellett az elküldött és fogadott üzenetek száma is gyakran használt hatékonysági jellemző.

5.2. Vezető választása

Ennek az alfejezetnek a témája az egyik legfontosabb döntési feladat, a *vezetőválasztás* (♣). Tegyük fel, hogy kezdetben minden processzor azonos állapotban van. A cél olyan állapot elérése, amelyben pontosan egy processzor a *vezető* (♣), a többi processzor pedig a *nem_vezető* állapotban van. A későbbiekben (főleg az algoritmusok leírásában) használjuk a rövidebb *vez*, illetve *nem_vez* jelölést is.

Számos feladat megoldásához szükség van a processzorok szimmetriájának megtörésére, és egy *vezető processzor* megválasztására. Ezt a feladatot LeLann fogalmazta meg 1977-ben.

Először megmutatjuk, hogy a vezetőválasztás bizonyos körülmények között *megoldhatatlan feladat*.

Azután algoritmusokat mutatunk be és elemzünk, amelyek gyűrűben, fában és általános hálózatban megoldják a vezetőválasztást.

5.2.1. Vezetőválasztás megoldhatatlansága gyűrűben

Legyen H egy p-processzoros gyűrű. Ha H-ban minden processzor azonos kezdeti állapotban van, akkor nincs mód ennek a kezdeti szimmetriának a megszüntetésére. Ezt az állítást formalizálja a következő tétel.

5.1. tétel. (*Vezetőválasztás gyűrűben.*) *Ha G egy egyirányú vagy kétirányú gyűrű, melyben a processzorok kezdeti állapota, állapotátmeneti függvénye és üzenet-előállító függvénye is azonos, akkor ebben a gyűrűben a vezetőválasztás nem oldható meg.*

Bizonyítás. Az állítást indirekt módon bizonyítjuk. Tegyük fel, hogy a P algoritmus megoldja a feladatot.

Feltehetjük, hogy a gyűrű minden processzorának csak egy kezdőállapota van (ha több van, közülük tetszőlegesen választva elérhetjük, hogy minden processzornak csak egy kezdőállapota legyen).

Az első lépésben minden processzor ugyanazt az üzenetet küldi szomszédjának (kétirányú gyűrűben mindkét szomszédjának), ezért a második lépésben a processzorok azonos új állapotba mennek át és azonos üzenet küldenek szomszédjuknak.

A lépések száma szerinti indukcióval adódik, hogy ha bármely processzor állapota *vezető*, akkor a többi processzor állapota is *vezető* lesz, ami nem biztosítja a vezetőválasztás egyértelműségét. ■

A gyakorlatban rendszerint nem azonos a processzorok kezdőállapota. A továbbiakban feltesszük, hogy minden processzornak egyedi *azonosítója* (♣) van, amely a többi processzortól megkülönbözteti.

5.2.2. Vezetőválasztás gyűrűben

Az első gyűrűs vezetőválasztó algoritmus LeLann nevéhez fűződik. A L L algoritmusra jellemző, hogy a szükséges üzenetek száma négyzetesen nő a processzorok számával. Chang és Roberts 1979-ben olyan javítást dolgoztak ki, amely legrosszabb esetben továbbra is négyzetes volt, de átlagos esetben már $O(n \lg n)$ lépésben megoldotta a vezetőválasztást. 1980-ban Hirschberg és Sinclair olyan megoldást talált, melyre a legrosszabb esetben is bizonyítani tudták az $O(n \lg n)$ felső korlátot. Igaz, míg a korábbi módszerek egyirányú gyűrűben is működtek, a H -S algoritmusnak kétirányú adatátviteli vonalakra van szüksége.

Az alsó korlátokkal kapcsolatos eredmények szerint a vezetőválasztást aszimptotikusan optimálisan is meg tudjuk oldani.

5.2.2.1. LeLann algoritmusa

A L L algoritmus megengedi, hogy a vezető az úgynevezett *kezdő processzorok* (♣) közül kerüljön ki. A kezdő processzorok halmazát K-val jelöljük. A processzoroknak nincs szüksége arra, hogy ismerjék a hálózat méretét.

Az algoritmus pszeudokódja a következő.

L L (K, A) *párhuzamos eljárás*

Számítási modell: egyirányú gyűrű

Bemenet: K (a kezdő processzorok indexeinek halmaza), $A[1 : n]$ (a
processzorok azonosítóinak tömbje,
amely különböző egész számokat tartalmaz)

Kimenet: i (a vezető processzor indexe)

```
P_i in parallel for 1 ≤ i ≤ n
    if i ∈ K then
            áll[i] := jelölt
            J_i := {i}
        else
            áll[i] := n_vez
P_i in parallel for 1 ≤ i ≤ n
    if áll[i] = jelölt then
                K¨ _i(i)
                while áll[i] ≠ vez
                    F _i(a)
                    J_i := J_i ∪ {a}
                    K¨ _i(a)
                if i = min{J_i} then
                        áll[i] := vez
                else
                        áll[i] := n_vez
    else
        while áll[i] ≠ vez
                    F _i(a)
```

```
20                                   K¨   i(a)
```

Az algoritmus szerint először az 1–6. sorokban beállítjuk a processzorok állapotát: *jelölt* lesz a kezdő processzorok állapota és *nem_jelölt* lesz a többi processzor kezdőállapota. A kezdő processzorok a jelöltek J_i halmazába beteszik saját azonosítójukat.

A 7–13. sorokban a kezdő processzorok addig fogadják és küldik az üzeneteket, amíg saját azonosítójukat – amely körbeért a gyűrűn – vissza nem kapják.

A 14–16. sorokban a legkisebb azonosítójú processzor *vez*-re, a többi kezdő processzor *n_vez*-re állítja a saját állapotát.

A nem kezdő processzorok szerepe az üzenetek továbbítása (17–21. sorok).

5.2. tétel. *A* L L *algoritmus egy egyirányú gyűrűn minden esetben $\Theta(p)$ lépésben és legrosszabb esetben $O(p^2)$ üzenetet küldve oldja meg a vezetőválasztást.*

Bizonyítás. Az 1–6. sorokban két lépésben beállítjuk *áll*$[i]$ és J_i értékét. A kezdő processzorok a p. lépésben visszakapják saját azonosítójukat (a többi kezdő processzor azonosítóját már korábban megkapták). Ekkor a legkisebb azonosítójú processzor állapota a 14–15. sor szerint *vezető* lesz, a többi kezdő processzor állapota pedig a 16. sorban *nem_vezető* lesz.

Mivel legfeljebb p különböző azonosító van és mindegyik p lépést tesz, ezért az elküldött és fogadott üzenetek száma $O(p^2)$. Mivel legrosszabb esetben minden processzor kezdő, ezért az üzenetek száma $W(p, \text{L L}) = \Theta(p^2)$. Mivel a kezdő folyamatok azonosítója p lépés alatt ér körbe a gyűrűn, ezért az algoritmus lépésszáma minden esetben $p = \Theta(p)$. ■

A L L algoritmus biztosítja, hogy a processzorok a megfelelő állapotba kerüljenek, de nem biztosítja azt, hogy a nem kezdő processzorok megálljanak. Ezt a megállást például úgy biztosíthatjuk, hogy a vezetőnek választott processzor körbeküld egy *értesítő üzenetet* (♣). Az így kiegészített algoritmusra is érvényes a $W(p,$ É ´-L - L $) = O(p)$ és $W_{ü}(p,$ É ´-L L $) = O(p^2)$

5.2.2.2. Chang és Roberts algoritmusa

Chang és Roberts azzal javították az előző algoritmust, hogy csökkentették a feleslegesen továbbküldött azonosítók számát: a kezdő processzorok csak a saját azonosítójuknál kisebb azonosítókat küldik tovább.

C -R (U, i) *párhuzamos eljárás*

Számítási modell: egyirányú gyűrű

Bemenet: $U = u_1,\ u_2,\ \ldots,\ u_p$ (a processzorok azonosítói – különböző egész számok)

Kimenet: i (a vezető processzor indexe)

P_i **in parallel for** $1 \leq i \leq n$
  if $i \in K$ **then**
      áll$[i]$:= *jelölt*
      $L_i := \{i\}$
    else
      áll$[i]$:= *nem_jelölt*
P_i **in parallel for** $1 \leq i \leq n$
  if *áll*$[i]$ = *jelölt* **then**
        K¨ $_i(i)$
        while $j \neq i$
          F $_i(a)$
          $L := L \cup \{j\}$
          K¨ $_i(a)$
        if $i = \min\{K\}$ **then**
            áll$[i]$:= *vez*

```
                    else áll[i] := n_vez
          else
             while áll(i) ≠ vez
                    F    i(j)
                    K¨   i(j)
```

5.3. tétel. *A* C -R *algoritmus* $\Theta(p)$ *lépéssel és legrosszabb esetben* $\Theta(p^2)$ *üzenettel oldja meg egyirányú gyűrűben a vezetőválasztást. Az algoritmus átlagos üzenetszáma* $O(p \lg p)$*.*

Bizonyítás. A legrosszabb esetre vonatkozó bizonyítás hasonló a L L algoritmusra vonatkozó bizonyításhoz.

Az átlagos üzenetszámmal kapcsolatban legyen s a legkisebb a p azonosító közül. p különböző azonosítónak $(p-1)!$ különböző ciklikus permutációja van. Adott ciklikus permutációban legyen a_i az az azonosító, amely i lépéssel halad s előtt.

Mivel az s azonosító minden permutációban p lépést tett meg, ezért a $(p-1)!$ ciklikus permutációban összesen $p(p-1)!$ lépést tett meg. Az a_i azonosítót legfeljebb i-szer kellett továbbítani, mivel eldobjuk, amikor eléri az s azonosítójú processzort. Legyen $A_{i,k}$ azoknak a ciklikus permutációknak a száma, amelyekben az a_i azonosítót pontosan k-szor kellett továbbítani. Ekkor az a_i azonosítót összesen

$$\sum_{k=1}^{i} k A_{i,k} \tag{5.2}$$

alkalommal kell továbbítani.

Ha a_i a legkisebb az $a_1,\ a_2,\ \ldots, a_i$ azonosítók között – ami $\frac{(p-1)!}{i}$ permutációban fordul elő – akkor az a_i azonosítót pontosan i-szer kell továbbítani, ezért

$$A_{i,i} = \frac{(p-1)!}{i} \ . \tag{5.3}$$

Ha az a_i azonosítót $k-1$ olyan azonosító követi, amelyek nagyobbak, mint a_i, akkor a_i-t legalább k-szor kell továbbítani (itt $k \leq i$). Azoknak a ciklikus permutációknak a száma, amelyekben a_i a legkisebb az $a_{i-k+1},\ a_{i-k+2},\ \ldots,\ a_i$ azonosítók között, $\frac{(p-1)!}{k}$. Ezért ha $k < i$, akkor az a_i azonosítót

$$\frac{(p-1)!}{k} - \frac{(p-1)!}{k+1} \tag{5.4}$$

permutációban kell pontosan k-szor továbbítani, és így

$$A_{i,k} = \frac{(p-1)!}{k(k+1)} \ \text{(ha } k < i). \tag{5.5}$$

Ezért az a_i azonosítót az összes ciklikus permutációban összesen

$$\sum_{k=1}^{i-1} k\left(\frac{(p-1)!}{k(k+1)}\right) + i\frac{1}{i}(p-1)! = (p-1)!\sum_{k=1}^{i}\frac{1}{k} \tag{5.6}$$

alkalommal kell továbbítani.

Ismert, hogy az egyenlőség jobboldalán lévő szumma a H_i *harmonikus szám*, amelyre

$$\sum_{i=1}^{m} H_i = (m+1)H_m - m. \tag{5.7}$$

Most összegezzük az s-től különböző i azonosítók által megtett lépések számát:

$$\sum_{i=1}^{p-1} \lfloor (p-1)!H_i \rfloor = (pH_{p-1} - (p-1))(p-1)!\ . \tag{5.8}$$

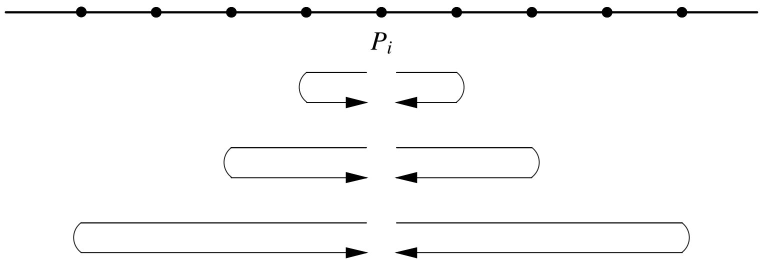
Mivel ez a lépésszám az összes ciklikus permutációhoz tartozik, ezért az átlag pH_p. Mivel $H_p = \ln p + O(1)$, azt kaptuk, hogy az átlag valóban $O(p \lg p)$. ■

Ha az azonosítók kezdeti permutációja kedvező – például $a_1 > a_2 > \ldots > a_p$ – akkor az a_j $(1 \leq j \leq p-1)$ azonosító csak egy lépést tesz meg, ezért $B_{ü}(p,$ C -R $) = O(p)$.

5.2.2.3. Hirschberg és Sinclair algoritmusa

Az eddig tárgyalt vezetőválasztó algoritmusok kevés lépést tesznek, de sok üzenetre van szükségük. Most egy olyan algoritmust mutatunk be, amelynek a korábbinál lényegesen kevesebb üzenetet igényel.

Hirschberg és Sinclair algoritmusa is a legnagyobb azonosítóval rendelkező folyamatot választja vezetőnek. Itt azonban az azonosítók nem körbejárják a gyűrűt, hanem bizonyos (egyre nagyobb) lépés megtétele után visszafordulnak. Ezt szemlélteti a 5.2. ábra.



5.2. ábra. A H -S algoritmus szemléltetése

5.4. tétel. *A* H -S *algoritmus üzenetszáma kétirányú gyűrűben* $W(p,$ H -S $) = O(p \lg p)$.

5.2.2.4. I ˝- algoritmus

Az eddigi vezetőválasztó algoritmusok az azonosítók összehasonlításával jutottak információhoz.

A következő I ˝- algoritmus nagyon kevés üzenetet használ. Az algoritmus szakaszokban működik, és minden szakasz p lépésből áll. A j-edik szakaszban csak j azonosítót lehet üzenetként elküldeni. Ha a P_i processzor azonosítója a_i, akkor ez a processzor az $1., 2., \ldots, (a_i - 1)$. szakaszban nem küld üzenetet. Ha a P_i processzor az első $a_i - 1$ szakaszban nem kap üzenetet, akkor az a_i-edik szakasz első lépésében elküldi szomszédjának a saját azonosítóját, és ez az azonosító körbemegy az egyirányú gyűrűn.

5.5. tétel. *Az* I ˝- *algoritmus egy p-processzoros egyirányú gyűrűben p üzenettel* $\Theta(pa_{min})$ *lépésben oldja meg a vezetőválasztást.*

Ennek a tételnek közvetlen következménye, hogy az I ˝- algoritmus üzenetszámát tekintve aszimptotikusan optimális.

5.2.2.5. Alsó korlát az üzenetszámra

Az összehasonlítás alapú vezetőválasztó algoritmusok üzenetszámára érvényes a következő alsó korlát.

5.6. tétel. *Ha a* $\mathcal{P}$ *algoritmus bármely p-processzoros gyűrűben vezetőt tud választani, akkor megadható p darab különböző azonosító olyan permutációja, amelyre az* $\mathcal{A}$ *algoritmus* $N_{ü}(p, P) = \Omega(p \lg p)$ *üzenetet küld.*

5.2.3. Vezetőválasztás fában

Az alábbi algoritmus fában megoldja a vezetőválasztást.

F´ - ˝(A, i) *párhuzamos eljárás*

Számítási modell: gyűrű

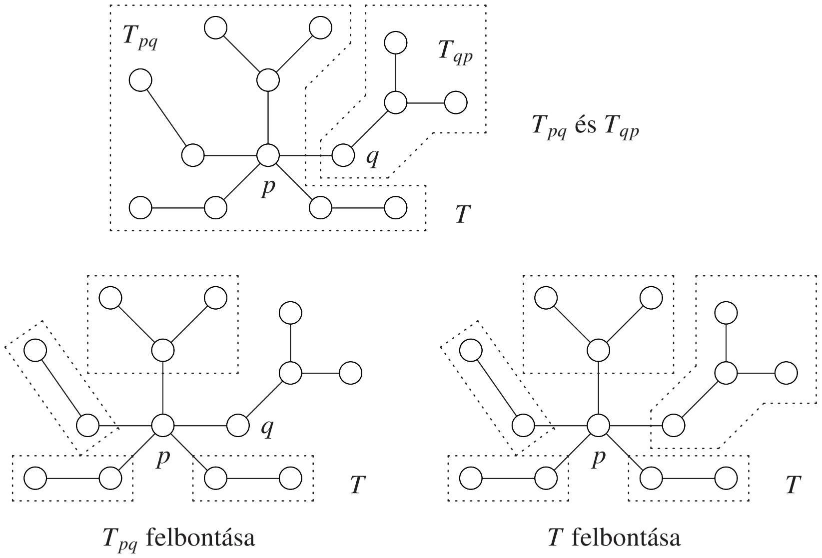
Bemenet: $A[1 : p]$ (a processzorok azonosítói, különböző egészek)

Kimenet: i (a vezető processzor indexe)

P_i **in parallel for** $1 \leq i \leq n$
    if $i \in K$ **then**
        $áll[i] :=$ *jelölt*

```
        else
            áll[i] := n_vez
```

Az algoritmus menetét illusztrálja az 5.3. ábra. Az ábra felső része a T fának a T_{pq} és T_{qp} részfákra bontását mutatja. Az ábra bal alsó része a T_{pq} fa felbontását mutatja.



5.3. ábra. T_{pq} részfái

5.7. tétel. *A* F´ - ˝ *algoritmus egy p-processzoros fában $O(p)$ üzenettel és $O(tmr)$ lépésben megoldja a vezetőválasztást.*

5.2.4. Vezetőválasztás általános hálózatban

Általános hálózatban először egy egyszerű üzenetterjesztő algoritmust, majd annak javított változatát mutatjuk be.

5.2.4.1. M - algoritmus

A M - algoritmus alapötlete, hogy a processzorok minden menetben elküldik szomszédaiknak az addig hozzájuk eljutott legnagyobb azonosítót.

5.8. tétel. *Ha egy tetszőleges H hálózat átmérője átm(H), akkor a* M - *algoritmus ebben a hálózatban legfeljebb átm(H) menetben a legnagyobb azonosítójú folyamatot vezetővé választja. Az elküldött üzenetek száma pedig $O(|E|tm)$.*

5.2.4.2. O - - algoritmus

Általános hálózatban is alkalmazható az a javítás, amit már a gyűrű esetében láttunk: a processzorok csak akkor küldenek tovább azonosítót, ha az új információt tartalmaz.

Ezzel ugyan a legrosszabb esetben szükséges üzenetek számának nagyságrendje változatlan marad, az üzenetek átlagos száma azonban lényegesen csökken.

5.9. tétel. *Ha egy tetszőleges H hálózat átmérője átm(H), akkor az* O - - *algoritmus ebben a hálózatban legfeljebb átm(H) menetben vezetővé választja a legnagyobb azonosítójú folyamatot.*

5.2.5. Alsó korlát az üzenetek számára

Az általános hálózatokban szükséges üzenetek számára vonatkozik a következő tétel.

5.10. tétel. *Ha H egy p processzort tartalmazó hálózat, akkor a vezetőválasztás ebben a hálózatban*

$$N(p) \geq pH_p \tag{5.9}$$

üzenetet igényel.

Ebből a tételből adódik a következő állítás.

5.11. következmény. *A* C -R *algoritmus üzenetszáma aszimptotikusan optimális.*

Bizonyítás. A 5.3. tétel szerint az algoritmus üzenetszámára

$$W(p) = O(p \lg p). \tag{5.10}$$

Mivel $H_p = \Theta(\lg p)$, így $\Theta(W(p)) = N(p)$. ■

5.3. Megegyezés

A következő döntési feladat a megegyezés. Tegyük fel, hogy kezdetben minden P_i processzor rendelkezik egy b_i bemeneti értékkel, és az a cél, hogy a processzorok azonos k kimenő értékre jussanak.

Ezt a problémát mind az üzenetek egy részének elvesztését, mind a processzorok hibáját megengedve is szokták vizsgálni.

A k-megegyezés (♣) problémája az egyszerű megegyezési probléma természetes általánosítása: a processzorok feladata az, hogy a bemenő értékek egy k-elemű részhalmazából válasszanak közösen elfogadott értéket.

5.3.1. Megegyezés vonalhibák esetében

A probléma lényegét jól tükrözi az *összehangolt támadási feladat* (♣). Eszerint tábornokok összehangolt támadást terveznek közös célpont ellen. A tábornokok hírnökök segítségével válthatnak üzenetet.

Feltesszük, hogy a tábornokok egy G irányítatlan (nem teljes) gráf csúcsaiban vannak, és az élek mentén küldhetnek üzenetet. Megbízható élekkel $G_{átm}$ lépésben minden tábornok teljes információval rendelkezik a többiek véleményéről, és a katonai akadémián tanultak alapján ugyanarra a döntésre juthatnak.

Ha azonban az élek meghibásodhatnak, ez az egyszerű gondolatmenet nem alkalmazható, a probléma nem oldható meg (ennek belátását meghagyjuk gyakorlatnak).

Hibás élek esetén csak az a reális célkitűzés, hogy megadott valószínűséggel jussanak a tábornokok közös véleményre. A problémának ez a változata már determinisztikus és véletlenített algoritmussal is kezelhető.

5.3.2. Megegyezés processzorhibák esetében

A processzorok működése során különböző *hibák* (♣) fordulhatnak elő. Az egyik a *megállási hiba* (♣), melyben a processzor bármely lépésben beszüntetheti működését. A másik a *bizánci hiba* (♣), melyben a processzorok a számukra megadott korlátokon (elvégezhető műveletek, felhasználható üzenetábécé) belül tetszőlegesen működhetnek.

Ennek a problémának egy egyszerű megoldását biztosítja a H - algoritmus. Ennek lényege, hogy a processzorok türelmesen terjesztik a tudomásukra jutott összes információt – és ha bizonyos ideig nem kapnak új információt, akkor az addig kapott üzenetek alapján döntenek.

Ha a processzorok értékelik is a beérkezett információt és csak a lényeges részt adják tovább, akkor az elküldendő üzenetek száma csökkenthető. Így jutunk az O - - algoritmushoz.

5.3.3. *k*-megegyezés

A *k*-megegyezési feladatnál a processzoroknak a bemeneti értékek *k*-elemű részhalmazából kell közösen elfogadott értéket választaniuk.

Ezt a feladatot például a M - algoritmussal lehet megoldani. Ennek lényege, hogy a processzorok karbantartják és terjesztik az addig kapott legkisebb értéket. Erről az algoritmusról belátható, hogy ha legfeljebb *h* processzor hibásodhat meg, akkor $\lfloor \frac{h}{k} + 1 \rfloor$ lépésben megoldja a feladatot.

A következő alsó korlát ismert.

5.12. tétel. *Ha $p \geq h + k + 1$, akkor minden algoritmusnak legalább $\lfloor \frac{h}{k} + 1 \rfloor$ lépésre van szüksége, hogy h hibás processzor esetén megoldja a k-megegyezési feladatot.*

5.3.4. Közelítő megegyezés

A *közelítő megegyezési problémánál* minden processzornak van egy valós kezdeti értéke és a processzorok valós értékeket küldenek egymásnak és egymástól kevéssé eltérő értékekben kell megegyezniük.

Megengedjük, hogy a processzorok bizáci hibákat kövessenek el.

3 feltételnek kell teljesednie.

A *befejeződési feltétel* szerint minden hibátlanul működő processzornak végül döntést kell hoznia.

Az *érvényességi feltétel szerint* a hibátlanul működő processzoroknak a hibátlan processzorok kezdeti értékeit tartalmazó (lehető legrövidebb) intervallumból vett értékkel kell megállniuk.

A *megegyezési feltétel* szerint akármely két hibátlanul működő processzor kimenő értéke legfeljebb egy előre adott ϵ értékkel térhet el egymástól.

Ismertek olyan algoritmusok, amelyek teljes hálózatban biztosítják a közelítő megegyezést, ha a hibás processzorok száma kisebb, mint az összes processzorok számának egy harmada.

5.4. Gyakorlatok

5-1. Elemezzük a L L és a C -R algoritmusokat.

(a) Adjuk meg az azonosítók olyan permutációját, amelyre az elküldött üzenetek száma $\Omega(n^2)$.

(b) Adjuk meg az azonosítók olyan permutációját, amelyre az elküldött üzenetek száma $O(n)$.

5-2. Módosítsuk az C -R algoritmust úgy, hogy az összes nem-vezető folyamat a *nem_vezető* kimenetet eredményezze, vagyis az összes folyamat végül is álljon meg. Adjuk meg a módosított algoritmus pszeudokódját.

5-3. Mutassuk meg, hogy a C -R algoritmus különböző induló időpontok mellett is helyesen működik. (Ehhez módosítsuk a kódot.)

5-4. Bizonyítsuk be a L L és a C -R algoritmusok helyességét.

5-5. Mutassuk meg, hogy a H -S algoritmus különböző induló időpontok mellett is helyesen működik. (Ehhez egy kicsit módosítsuk a pszeudokódot.)

5-6. Tegyük fel, hogy a H -S algoritmust úgy módosítjuk, hogy kettő-hatványok helyett egymás utáni k-hatványokat használunk az utak hosszára ($k > 2$). Elemezzük a módosított algoritmus lépésszámát és kommunikációs bonyolultságát úgy, mint az eredeti H -S algoritmusnál. Hasonlítsuk össze az eredményeket.

5-7. Tekintsük a H -S algoritmus olyan módosított változatát, ahol a processzorok mindkét irány helyett csak az egyik irányba küldhetnek üzeneteket.

(a) Mutassuk meg, hogy a könyvben megadott algoritmus legkézenfekvőbb módosítása nem eredményez $O(n \log n)$ üzenetszámot. Adjunk felső korlátot az üzenetszámra.

(b) Módosítsuk úgy az algoritmust, hogy az üzenetszáma $O(n \log n)$ legyen.

5-8. Tervezzünk egyirányú gyűrűben olyan vezetőválasztó algoritmust, amely nem ismeri a gyűrű méretét és legrosszabb esetben is csak $O(n \log n)$ számú üzenetet használ. Az algoritmus az azonosítókra kizárólag az összehasonlítás műveletet használhatja.

5-9. Adjunk a menetek számára vonatkozó minél jobb *alsó* korlátot valamely n méretű gyűrű vezető folyamat kiválasztásos algoritmusának legrosszabb esetére. A feltevéseket körültekintően fogalmazzuk meg.

5-10. Adjuk meg az $n = 16$ csúcsú bitfordító gyűrű pontos leírását.

5-11. Bizonyítsuk be, hogy az $n = 2^k$ méretű bitfordító gyűrű minden $k \in \mathbb{N}$ esetén $\frac{1}{2}$-szimmetrikus.

5-12. Tervezzünk c-szimmetrikus gyűrűt nem kettő-hatvány számú csúcs esetén valamilyen $c > 0$ értékre.

5-13. Valamely szinkron gyűrű esetén tekintsük a vezető folyamat kiválasztásának problémáját, ahol minden folyamat ismeri a gyűrű n méretét és a processzoroknak nincs azonosítójuk. Adjunk a probléma megoldására *véletlenített algoritmust*, vagyis olyat, ahol a processzorok kódjuk determinisztikus végrehajtásán kívül véletlen választással is élhetnek. A helyes működést kielégítő tulajdonságokat óvatosan fogalmazzuk meg. Például az egyedi vezető folyamat kiválasztása biztosan garantált-e vagy valamilyen kis valószínűséggel elképzelhető, hogy ez nem történik meg? Mennyi lesz az algoritmus lépésszáma és üzenetszáma?

5-14. Tekintsünk valamilyen ismeretlen n méretű kétirányú gyűrűt, ahol a processzoroknak van egyedi azonosítójuk. Adjunk az üzenetek számára vonatkozó alsó és felső korlátot olyan összehasonlítás alapú algoritmus esetén, ahol minden processzor mod 2 számolja ki n-et.

5-15. A M - algoritmusban használt üzenetek *átm*$|E|$ száma $O(n^3)$. Adjunk meg olyan irányított gráfokat, amelyekre az *átm*$|E|$ szorzat $\Omega(n^3)$, vagy mutassuk meg, hogy nincs ilyen irányított gráf.

5-16. Az O - - algoritmus által elküldött üzenetek számára adjunk a $O(n^3)$-nál kisebb felső korlátot vagy mutassuk meg, hogy a korlát aszimptotikusan éles.

5-17. Elemezzük a vezetőválasztás lépésszámát és üzenetszámát, feltéve, hogy néhány szomszédos csúcs között kétirányú kommunikációt is megengedünk.

5-18. Tervezzünk egy vezetőválasztó algoritmust egy olyan erősen összefüggő irányított hálózatban, amelyben a processzoroknak van egyedi azonosítójuk.

(a) Tegyük fel, hogy a kommunikáció a szomszédos csúcsok között kétirányú.

(b) Ne alkalmazzuk az előző feltevést.

5-19. Adjunk algoritmust a csúcsok számának megállapítására egy olyan erősen összefüggő irányított hálózatban, amelyben a processzoroknak van egyedi azonosítójuk.

(a) Tegyük fel, hogy a kommunikáció a szomszédos csúcsok között kétirányú.

(b) Ne alkalmazzuk az előző feltevést.

5-20. Adjunk algoritmust az élek számának megállapítására egy olyan erősen összefüggő irányított hálózatban, amelyben a processzoroknak van egyedi azonosítójuk.

(a) Tegyük fel, hogy a kommunikáció a szomszédos csúcs közt kétirányú.

(b) Ne alkalmazzuk az előző feltevést.

5-21. Tegyük fel, hogy egy láncban minden P_i processzor meg tudja különböztetni a bal oldalát a jobb oldalától, és ismeri azt is, hogy ő maga végpont-e vagy sem. Tegyük fel, hogy minden processzor kezdetben egy nagyon nagy a_i egész értékkel rendelkezik, és azt, hogy az ilyen értékekből egy adott időpillanatban csak adott számút tarthatunk nyilván a memóriában. Tervezzük meg azt az ezen értékeket sorba rendező algoritmust, amelyben az egyes P_i processzorok által előállított o_i kimeneti értékek összeszorzott halmaza megegyezik az a_i bemeneti értékek összeszorzott halmazával, és $o_1 \leq \ldots \leq o_n$. Próbáljuk meg előállítani mind az üzenetek, mind a menetek száma tekintetében a leghatékonyabb algoritmust.

5-22. Mutassuk meg, hogy az összehangolt támadási probléma (determinisztikus változatának) megoldása bármely nem triviális, összefüggő gráf esetében magában foglalja a probléma megoldását arra az egyszerű, két pontból álló gráfra, mely egy éllel van összekötve. (Ebből következik, hogy a probléma megoldhatatlan tetszőleges, nem triviális gráf esetében.)

5-23. Tekintsük a (determinisztikus) összehangolt támadási probléma következő változatát. Tegyük fel, hogy a hálózat $n > 2$ résztvevőből álló teljes gráf. A befejezési és érvényességi feltételek az 5.3. alfejezetben leírtakkal azonosak. A megegyezési feltételt azonban gyengítjük: „Ha van olyan a folyamatok között, amelyik döntése 1, akkor legalább kettőnek 1-est kell döntenie." (Azaz szeretnénk kizárni azt az esetet, amikor egy tábornok magányosan támad, de megengedjük azt, hogy két vagy több tábornok együtt támadjon.) Vajon ez a probléma megoldható, vagy nem?

5-24. Tekintsük az összehangolt támadási problémát vonalhibák esetében arra az egyszerű esetre, amikor két folyamat egy éllel van összekötve. Tegyük fel, hogy a processzorok működése determinisztikus, de az üzenetrendszer véletlenített abban az értelemben, hogy mindegyik üzenetnek van egy független q valószínűségi értéke ($0 < q < 1$), ami annak a valószínűségét adja meg, hogy az üzenet sikeresen megérkezik. (Ahogy általában, most is megengedjük, hogy a folyamatok menetenként csak egy üzenetet küldjenek.) Tervezzünk ezekkel a beállításokkal olyan algoritmust, mely rögzített r számú meneten belül befejeződik, a megegyezés hiányának valószínűsége legfeljebb ϵ, és ehhez hasonlóan az érvényességi feltétel megsértésének valószínűsége is legfeljebb ϵ. A lehető legkisebb ϵ érték elérésére törekedjünk.

5-25. Az előző gyakorlat kikötései szerinti modellben adjunk alsó korlátot az ϵ értékére, bizonyítsuk be, hogy ez az elérhető legalacsonyabb érték.

5-26. Általánosítsuk az összehangolt támadási probléma véletlenített változatát úgy, hogy megengedjük ϵ valószínűséggel mind az érvényességi, mind a megegyezési szabályok megsértését. Írjuk át a V´ ´ - ´ ´ algoritmust úgy, hogy a módosított feltételek mellett elérje a lehető legkisebb ϵ értéket. Végezzünk elemzést.

5-27. Általánosítsuk a V´ ´ - ´ ´ algoritmust és az elemzését az általános irányítatlan gráfokra.

5-28. Mi történne a fejezetben tárgyalt, véletlenített környezettel kapcsolatos eredményekkel, ha az ellenfél kommunikációs mintája nem lenne előre rögzítve, mint ahogy eddig feltettük, hanem az ellenfél közvetlen irányítással határozhatná meg azt. Pontosabban szólva, tegyük fel, hogy az ellenfél képes arra, hogy megvizsgálja a végrehajtási sorozatot bármely k-adik menettől visszafelé a kezdetig, mielőtt döntene, hogy a k-adik menetbeli üzenetek közül melyek legyenek kézbesítve.

(a) Milyen ϵ korlát garantálható a V´ ´ - ´ ´ algoritmus esetében a megegyezés hiányára, ilyen közvetlen irányításra képes ellenfelek esetében?

(b) Adhatunk-e valamilyen érdekes alsó korlátot az elérhető ϵ értékekre?

5-29. Bizonyítsuk be, hogy tetszőleges olyan algoritmus, mely megoldja a bizánci megegyezés problémát, megoldja a megegyezési problémát megállási hibák esetében is, ha a megállási hiba modellben úgy módosítjuk az érvényességi feltételt, hogy csak a hibamentes folyamatok megegyezését követeljük meg.

5-30. Bizonyítsuk be, hogy tetszőleges olyan algoritmus, mely megoldja a bizánci megegyezés problémát és amelyben a hibátlan folyamatok mindig egyszerre, ugyanazon menetben hoznak döntést, megoldja a megegyezési problémát megállási hiba modellben is.

5-31. Kövessük nyomon a H - algoritmus végrehajtását négy folyamattal és két hibával, melyben a folyamatok kezdőértékei rendre az 1, 0, 0, 0 értékek. Tegyük fel, hogy P_1 és P_2 folyamatok hibásak, P_1 az első menetben lesz hibás, miután egyedül a P_2 folyamatnak elküldte az üzenetet, P_2 pedig a második menetben lesz hibás, P_1-nek és P_3-nak küld üzenetet, viszont P_4-nek nem.

5-32. Tekintsük a H - algoritmust f hibára. Tegyük fel, hogy az algoritmus $f + 1$ menet helyett csak f menetben fut, ugyanazzal a döntési értékkel. Találjunk egy olyan végrehajtási sorozatot, mely megsérti a helyességi feltételeket.

5-33. Legfeljebb mennyi lehet a hibamentes folyamatok által hozott, egymástól különböző döntési értékek darabszáma, ha a H - algoritmus $f + 1$ menet helyett csak f menetben fut.

5-34.

(a) Találjunk egy másik lehetséges, helyesen működő döntési szabályt a H - algoritmusban, amelyik eltér szövegben megadottól.

(b) Adjunk pontos jellemzést azon döntési szabályok halmazáról, amelyek helyesen működnek.

5-35. Terjesszük ki a H - algoritmust, a helyesség bizonyítását, és az elemzést, tetszőleges összefüggő gráfokra.

5-36. Készítsük el az O - - algoritmus kódját.

5-37. Tekintsük a következő egyszerű algoritmust a megállási hibák mellett történő megegyezésre, egy adott V értékhalmaz esetében. Legyen mindegyik processzornak egy *min-érték* változója, melyet induláskor a saját kezdeti értékére állít be. Az $f+1$ menet mindegyikében a processzorok közreadják *min_érték* változójuk értékét, majd újra beállítják úgy, hogy a minimuma legyen a *min_érték* változó eredeti értékének, valamint az üzenetekben kapott értékeknek. Végül a processzor döntési értéke *min_érték* lesz. Készítsük el ennek az algoritmusnak a kódját és bizonyítsuk be (vagy direkt módon, vagy szimulációval), hogy helyesen működik.

5.5. Feladatok

5-1. Vezetőválasztás négyzeten

Bizonyítsuk be, hogy négyzeten $O(n \log n)$ idő alatt megoldható a vezető választása.

5-2. Vezetőválasztás tóruszon

Bizonyítsuk be, hogy tóruszban $O(n \log n)$ idő alatt megoldható a vezetőválasztás.

5-3. Vezetőválasztás hiperkockán

Bizonyítsuk be, hogy hiperkockán $O(n \log n)$ idő alatt megoldható a vezetőválasztás.

5-4. Nem összehasonlítás alapú vezetőválasztás

Az anyagban csak összehasonlítás alapú vezetőválasztó algoritmusokat tárgyaltunk. Vizsgáljunk meg néhány más típusú algoritmust is.

(a) Írjuk meg az I ˝- algoritmus pszeudokódját.

(b) Módosítsuk úgy az I ˝- algoritmust, hogy hozzávett üzenetek árán fázisonként egyetlen azonosító helyett k darab azonosító továbbküldésének engedélyezésével csökkenjen a lépésszám. Bizonyítsuk be az algoritmus helyességét és elemezzük bonyolultságát.

(c) Adjuk meg a V´ ´- ´ algoritmus pszeudokódját.

(d) Mutassuk meg, hogy ha a processzorok különböző időpontokban ébredhetnek fel, a V´ ´- ´ algoritmus üzenetszáma nem szükségszerűen $O(n)$.

5-5. Harmonikus szám

Bizonyítsuk be a harmonikus számok alábbi tulajdonságait:

$$\sum_{i=1}^{n} H_i = (n+1)H_n - n \quad (\text{ha } n \geq 1) \tag{5.11}$$

és

$$\ln(n+1) < H_n < 1 + \ln(n+1) \quad (\text{ha } n \geq 1). \tag{5.12}$$

5-6. C -R algoritmus

- Ha minden processzor kezdő processzor, legjobb esetben hány üzenetet továbbít a C -R algoritmus?
- Ha pontosan s kezdő processzor van, amelyek egyforma valószínűséggel lesznek vezetők, akkor mennyi lesz az algoritmus átlagos *kommunikációs bonyolultsága* (üzeneteinek száma)?

5-7. Vezetőválasztás síkhálózatokban

Mutassuk meg, hogy ha egy hálózat síkba rajzolható, akkor tervezhető rá olyan vezetőválasztó algoritmus, amelynek $O(p \lg p)$ üzenetre van szüksége.

5-8. Véglegesítés

Tervezzünk egy algoritmust, amely a véglegesítési problémát az erős befejezési feltétellel megoldja. Elérhető-e egyidejűleg, hogy a menetek száma legrosszabb esetben $n+k$ legyen (ahol k konstans), a hibamentes esetben a döntéshez és megálláshoz szükséges menetek száma egy kis konstans legyen és a hibamentes esetben alacsony legyen az üzenetszám?

5-9. Bizánci megegyezés

Tervezzünk a bizánci megegyezés megoldására *egyszerű* $f + 1$ menetes algoritmust, melyhez csak $3f + 1$ processzor szükséges, és üzenetszáma polinomiális.

6. Hagyományos és elektronikus irodalom

6.1. Megjegyzések az 1. fejezethez

A párhuzamos algoritmusok alapját képező soros algoritmusok magyar nyelvű szakirodalma jelentős. A leggazdagabb anyagot Knuth monográfiája [198, 199, 200, 202], valamint Cormen, Leiserson és Rivest [53] könyve – melyeket több mint tíz nyelvre lefordítottak – tartalmazza.

Több témában hasznos anyagot tartalmaznak – ugyancsak magyarul – Aho, Hopcroft és Ullman [3, 4], Artiaga és Davis [16], Kása Zoltán [179] Lawler [229], Lovász és Gács [241], Marton és Fehérvári [248], Papadimitriou [282], Rónyai, Ivanyos és Szabó [314], Trahtenbrot [365], valamint Wirth [388] könyvei.

Az angol és német nyelvű könyvek közül elsősorban Berman és Paul [32], Cormen, Leiserson, Rivest és Stein ([53] új, bővített kiadása), Mehlhorn [250, 251, 252] valamint az Atallah [17], Gonnet és Baeza-Yates [116], Gruska [119], Ralston, Reilly és Hemmendinger [306], valamint a van Leewen [369] által szerkesztett enciklopédiákat ajánljuk.

Kiegészítésképpen érdemes elolvasni Baase [21], Baase és Van Gelder [22], Brassard és Bratley [36], Gibbons és Rytter [109], Greenflaw, Hoover, Ruzzo [118], Jájá [169], Miller és Boxer [254], Selim [330], Skiena [339], Valiente [368], valamint Wilf [385] könyveit.

Elektronikus formában több algoritmusgyűjtemény elérhető, például a CALGO [40], LEDA [253], NetLib [264] és XTANGO [389].

A párhuzamos algoritmusok témakörében az első lényeges gondolatok Neumann Jánosnak a negyvenes évek végén és az ötvenes évek elején elhangzott előadásaiban és azóta megjelent dolgozataiban [266, 267, 268, 270] találhatók. Ezekben a sokprocesszoros rendszereken belül való munkamegosztás problémáira is felhívta a figyelmet. Azt is igazolta, hogy kellő többszörözéssel védekezhetünk a sokkomponensű rendszerek elemeinek bizonyos mértékű meghibásodásával szemben. Ezen gondolatok egy része magyarul is hozzáférhető Drommerné Takács Viola [72] és Neumann János [268] könyvében.

A *párhuzamos algoritmusok* magyar nyelvű irodalma még kicsi. Figyelemre méltó anyag Lovász László és Gács Péter 1978-ban megjelent, az algoritmusokat népszerűsítő könyvében [241] a párhuzamos számítások lehetőségeiről szóló fejezet. N. A. Lynch [245] könyve részletesen tárgyalja az *osztott rendszerekkel* (elsősorban az aszinkron hálózatokkal) kapcsolatos problémák egy részét.

Az algoritmusokkal kapcsolatos *szakkifejezések* egy része is megtalálható a Frey Tamás és Szelezsán János által a hőskorban szerkesztett könyvekben [96, 97], valamint a Horváth László és Pirkó József által szerkesztett lexikonokban [138, 139, 140]. Ezeket egészítjük ki a könyvünk *Angol kifejezések* és *Magyar kifejezések* című részeiben lévő szótárakkal.

A könyv anyagának elmélyítéséhez elsősorban a következő műveket ajánljuk.

A *függvények növekedésével* kapcsolatos alapfogalmakat Knuth cikkéből [194, 201], Almasi és Gottlieb hangulatos leírásából [6], valamint Schöning könyvéből [328] vettük át. A *Hanoi tornyaira* vonatkozó feladat Lucastól [244] származik. A feladat leírása magyarul is hozzáférhető, például Knuth, Graham és Patashnik [195] könyvében. A Föld életkorára vonatkozó becslés az Officina Nova kiadó világatlaszából származik [278].

A *matematikai alapok* ismétléséhez Andrásfai Béla [13], Demetrovics János, Denev és Pavlov [64], Dringó László és Kátai Imre [71], valamint Láng Csabáné és Gonda János [115, 225, 226] tankönyveit ajánljuk.

Amdahl törvénye a [8], Gustafson törvénye pedig a [120] konferenciakötetben jelent meg.

Az alkalmazott *pszeudokódot* Cormen, Leiserson, Rivest és Stein [54], valamint Berman és Paul [32] Pascal-alapú pszeudokódja és a Stroustrup által kifejlesztett C++ nyelv [349] ötvözésével állítottuk össze.

A *hatékonysági mértékeket* többek között Horowitz, Sahni és Rajasekaran [137], Lynch [245], valamint Roosta [315] és Tel [364] alapján definiáltuk. A lépésszámfüggvények polinomiális és exponenciális osztályokra való bontásával a bonyolultságelmélet *bibliáját*, Garey és Johnson könyvét [106] követtük.

A *számítási modelleket* illetően főleg Cormen, Leiserson, Rivest és Stein [54], valamint Berman és Paul [32] műveire támaszkodtunk.

Sima, Fountain és Kacsuk könyve [335] részletesen tárgyalja a hagyományos és a modern rendszerek *felépítését*. A könyvhöz jó kiegészítés a *számítási modellekkel* foglalkozó első fejezet kézirata [334], amely lényegesen bővebb a nyomtatásban megjelent változatnál. A párhuzamos rendszerek *hardver- és szoftverproblémáival* kapcsolatban gazdag anyag van például Almasi és Gottlieb [6], Amestoy [9], Kacsuk Péter és Kotsis Gabrielle [177], Kacsuk Péter, Kranzlmüller, Németh Zsolt és Volkert [178], valamint Leighton [233, 234] műveiben.

Flynn [85] 1966-ban publikálta *osztályozási rendszerét*.

Az ILLIAC IV számítógéppel foglalkozik Barnes [27] cikke.

A *párhuzamos számítógépek* előfutárának tekinthető D. H. Lehmernek [231, 232] a harmincas években épített mechanikus szerkezetét, amely párhuzamos műveletek elvégzésére is képes volt. Lehmer például a $2^{93}+1$ tényezőkre bontására használta fel a szerkezetet.

A *rekurzív algoritmusokkal és rekurzív egyenletekkel* kapcsolatban hasznos magyarul Cormen, Leiserson és Rivest [53], Knuth, Graham és Patashnik [195], angolul pedig Berman és Paul [32] Greene és Knuth [117], Purdom és Brown [298], Sedgewick és Flajolet [329], valamint Wilf [384] műve.

A *rekurziótétel* megtalálható például Halmos [124] könyvében.

A *véletlenített algoritmusokról* szóló alfejezethez Bollobás [34], Erdős és Spencer [75], Hofri [134], Motwani és Raghavan [260], Mayr, Prömel és Steiger [249], Pardalos és Rajasekaran [284], valamint Spencer [342] könyveit ajánljuk.

A Csernov-egyenlőtlenségeket Csernov [45] 1952-ben publikálta.

A *valószínűségszámítás* magyar nyelvű szakirodalmához tartoznak Feller [79], Prékopa [296] és Rényi [312] könyvei.

Az *anomáliára* példák szerepelnek Coffman [48], Iványi és Szmeljánszkij [168], valamint Roosta [315] könyvében, továbbá Fornai Péter és Iványi Antal [87], valamint Lai és Sahni [224].

Alsó korlátra vonatkozó eredmények találhatók például Berman és Paul [32], valamint Lynch [245] könyvében.

A párhuzamos és osztott rendszerek számos részkérdéséről tartalmaznak hasznos anyagot Akl [5], Gibbons és Rytter [109], Heath, Renade és Schreiber [128], Jájá [169], Leopold [236] és Zomaya [390, 391] könyvei.

A *gyakorlatok és feladatok* részben a már említett szakkönyvekből származnak, részben oktató- és kutatómunkánk során gyűltek össze. Érdemes meglátogatni Fekete István és Hunyadvári László [148], Járai Antal [172], Katona Gyula [182] magyar nyelvű – soros algoritmusokkal kapcsolatos – elektronikus feladatgyűjteményét. Nagyon hasznos Sussman oktatói kézikönyve, amely [53]-hez készült, továbbá Hecker [129] és Winkler [387] feladatsorozatai. Értékes oktatási segédeszköz Cormen, Lee és Lin oktatói segédkönyve [52], amely Cormen, Leiserson, Rivest és Stein könyvéhez [54] készült. Párhuzamos programokat tartalmaznak Rajasekaran [303] és Schreiner [340] honlapjai.

A további megjegyzések a gyakorlatokra és feladatokra vonatkoznak. Az első feladat alapja Cormen, Leiserson és Rivest [54], a második feladaté Roosta könyve [315], a harmadik Iványi Annától [153] származik. A negyedik feladathoz Coffman könyvének [48] Fornai Péter és Iványi Antal [87], valamint Lai és Sahni cikkének [224], az ötödikhez Berman és Paul könyvének [32], végül a hatodikhoz Iványi Antal cikkeinek [160, 165], valamint Kovács Gábor Zsolt és Pataki Norbert diákköri dolgozatának [210] elolvasása nyújthat segítséget.

A *kétdimenziós tökéletes mátrixokkal* kapcsolatban nagy anyag található D. E. Knuth monográfiájának elektronikus negyedik kötetében [202]. További algoritmusok találhatók a Belényesi Viktor és Németh Cs. Dávid diákköri dolgozatában [28], Ferenczi és Kása Zoltán [80], valamint Iványi Antal és Tóth Zoltán cikkeiben [155, 157, 158, 159].

Háromdimenziós tökéletes mátrixok előállítási algoritmusait tartalmazza Iványi Antal [159] 1990-ben megjent cikke, valamint Horváth Márk és Iványi Antal [141] friss technikai riportja.

A *bináris sorozatok* feldolgozásával kapcsolatos algoritmusok Iványi Antal és Kátai Imre [163, 164], valamint Iványi Antal és Pergel József [167] dolgozataiból származnak.

A *foksorozatokkal* kapcsolatosak Iványi Antal [160], Kemnitz és Dulff [186], Kovács Gábor Zsolt és Pataki Norbert [210], Moon [258], Narayana és Bent [261], Pécsy Gábor és Szűcs László [288], Siklósi Bence [332], valamint Szadovszkij és Szadovszkij [320] művei.

A *ládapakolással* kapcsolatban Coffman, Csirik János és Woeginger összefoglalóját [286], Coffman, Galambos Gábor, Martello és Vigo [50], Csirik János, Galambos Gábor, Frenk és Rinnoy Kan [57], Csirik János és Imreh Balázs [58], Csirik János és Johnson [59] cikkét, valamint Iványi Antal dolgozatait [154, 156] említjük.

6.2. Megjegyzések a 2. fejezethez

A *prefixszámítás aszinkron* megoldására például Horváth Zoltán cikkében [142] és elektronikus kéziratában [144], valamint Kozsik Tamás technikai riportjában [217] található algoritmus.

Nehéz lenne a *gráfelmélet* elméleti és gyakorlati jelentőségét túlbecsülni, ezért nem meglepő, hogy számos magyar nyelvű könyv szól a gráfokról: Andrásfai Béla [10, 11, 12], Busacker és Saaty [39], Cseke Vilmos [55], Kaufman [184], Katona Y. Gyula, Recski András és Szabó Csaba [183], Ore [280], Recski András [308] művei. Megemlítjük Kása Zoltán elektronikus feladatgyűjteményét [180] is.

Az idegen nyelvű könyvek közül Berge [30], Bollobás Béla [34], Diestel [67], Jungnickel [175] műveit említjük

A munkaoptimális véletlenített kereső algoritmust Floyd és Rivest [84] publikálta 1975-ben.

Preparata algoritmusa 1978-ban vált ismertté [297].

Reischuk munkahatékony rendező algoritmusa 1985 óta ismert [311].

A feladatok forrása Horowitz, Sahni és Rajasekaran [137], Berman és Paul [32], valamint Tel [364] könyve.

6.3. Megjegyzések a 3. fejezethez

Leighton könyve [233] a rácsalgoritmusokról szóló monográfia.

Véletlenített csomagirányítási algoritmust javasolt Rajasekaran és Tsantilas 1992-ben [302].

Aszimptotikusan optimális csomagirányítási algoritmust ismertet Nassimi és Sahni 1982-ben [263].

A konvex burok számításáról összefoglalás található Sack és Urrutia [322] kézikönyvében.

A feladatok megoldásához Berman és Paul [32], valamint Horowitz, Sahni és Rajasekaran könyve [137] a leghasznosabb.

6.4. Megjegyzések a 4. fejezethez

Leighton két könyve [233, 234], valamint Ranka és Sahni műve [307] jelentős anyagot tartalmaznak a hiperkockákkal kapcsolatban.

A feladatok többségének forrása Cormen, Leiserson, Rivest és Stein [53], valamint Horowitz, Sahni és Rajasekaran [137] könyve.

6.5. Megjegyzések az 5. fejezethez

Ehhez a fejezethez elsősorban N. A. Lynch magyarul is megjelent könyvét [245], Tanenbaum és van Steen [363], valamint Tel [364] friss monográfiáját említjük.

A magyarul is hozzáférhető anyagot csak röviden ismertetjük és kiegészítjük más algoritmusok elemzésével.

Le Lann 1977-ben egy konferencián [235] fogalmazta meg és oldotta meg a *vezetőválasztási* problémát.

Chang és Roberts két év múlva a [44] cikkben javasoltak átlagosan kevesebb üzenetet igénylő módszert. Peterson [289] cikke újabb 3 év múlva jelent meg. Hirschberg és Sinclair [132] 1980-ban javasolt $O(p \lg p)$ üzenetszámú algoritmust kétirányú gyűrűk vezetőjének megválasztására. Később Peterson [290] és Dolev, Klawe, Rodeh [68] egymástól függetlenül egyirányú gyűrűben is megoldották $O(p \lg p)$ üzenettel a vezetőválasztást.

Pachl, Korach és Rotem [281] 1984-ben mutatták meg, hogy minden gyűrűben átlagosan $\Omega(p \lg p)$ üzenetre van szükség a vezetőválasztáshoz. 1988-ban Bodlaender [33] bebizonyította, hogy kétirányú gyűrűben legrosszabb esetben legalább $0.34 p \lg p$ üzenetre van szükség.

A fejezet feladatai részben Lynch [245] és Tel könyvéből származnak [364].

A terjedelem szabta korlátok miatt a témakör több lényeges területével nem foglalkoztunk. Ezeket az ACM (*Association for Computing Machinery*) és az IEEE (*Institute of Electrical and Electronics Engineers*) által 2001-ben összeállított *Informatikai Tanterv*, a *Steelman Report* [151] alapján csoportosítjuk.

Ez a tanterv az informatikai ismeretek 14 különböző területét különbözteti meg (zárójelben megadjuk az ajánlott minimális előadási óraszámot, valamint a rövid és teljes angol nevet): *diszkrét matematika* (43 óra, **DS** = Discrete Structures), *programozás alapjai* (38 óra, **PF** = Programming Fundamentals), *algoritmusok és bonyolultság* (31 óra, **AL** = Algorithms and Complexity), *programozási nyelvek* (21 óra, **PL** = Programming Languages), *számítógépek felépítése* (18 óra, **AR** = Architecture and Organization), *operációs rendszerek* (18 óra, **OS** = Operating Systems), *hálózatok* (15 óra, **NC** = Net-Centric Computing), *ember-gép kapcsolat* (8 óra, **HC** = Human-Computer Interaction), *grafika* (3 óra, **GV** = Graphics and Visual Computing), *mesterséges intelligencia* (10 óra, **IS** = Intelligent Systems), *információkezelés* (10 óra, **IM** = Information Management), *szoftvertechnológia* (31 óra, **SE** = Software Engineering), *számítógép és társadalom* (16 óra, **SP** = Social and Professional Issues), *kiszámításelmélet* (0 óra, **CN** = Computational Science).

Az ismereteknek ebben a rendszerezésében könyvünk témája elsősorban az *algoritmusok* témakör 11 altémája közül kettő: a *párhuzamos algoritmusok* és az *osztott algoritmusok*.

1. Diszkrét matematika (DS). A *Steelman Report* a függvények, relációk és halmazok egyszerű tulajdonságait, logika elemeit, alapvető bizonyítási és leszámlálási módszereket, valamint a valószínűségszámítás alapfogalmait sorolja. Ajánlott irodalom magyarul Andrásfai Béla [13], Bagyinszki János és György Anna [24], Demetrovics János, Denev és Pavlov [64], Dringó László és Kátai Imre [71], Gavrilov és Szapozsenko [107], Gonda János és Láng Zsuzsa [225, 226, 115], valamint Járai Antal [172] elektr.

A legfontosabb matematikai képleteket és definíciókat tartalmazza Bronstein, Szemgyajev et al. [37] és Obádovics J. Gyula [?] könyve. A matematika tanításának pszichológiájába ad betekintést Klein Sándor [192] műve.

Jó angol nyelvű főiskolai könyvek Dossey, Otto, Spence és Eynden [70], valamint Prather [295] művei, szórakoztató Vilenkin könyve [375] a kombinatorikáról.

Komolyabb *matematikai alapokat* nyújtanak a logika területéről magyarul Flach [82], Pásztorné Varga Katalin és Várterész Magdolna [287], valamit Ruzsa Imre [318], lineáris algebráról Freud Róbert [94], számelméletről Freud Róbert és Gyarmati Edit [95], kombinatorikáról pedig Lovász László [239].

Angolul *algebrai algoritmusokról* Pethő Attila [291] és Winkler [386], kombinatorikáról pedig Erdős Pál [74], valamint Lovász László [239] adnak részletes ismereteket. Nagy anyagot tartalmaz a leszámlálás módszereiről Stanley [344] kétkötetes monográfiája.

Konkrét *kombinatorikai algoritmusokat* (és programokat) tartalmaz Nijenhuis és Wilf [271] könyve, valamint Knuth elektronikus kézirata [202] és elektronikus formában is elérhető könyve [196].

2. Programozás alapjai (PF). Ide a legfontosabb programkonstrukciók, feladatok specifikációja, egyszerű adatszerkezetek, rekurzió tartozik.

Ajánlott irodalom: magyarul Fóthi Ákos [89], angolul Baase és van Gelden [21, 22], oroszul pedig Iványi Antal és Szmeljánszkij [168] könyvei.

A szakirodalom értékes részét képezi azoknak a szoftver eszközöknek (*Mathematica, Matlab, Maple*) a leírása, melyek segítségével kényelmesen programozhatók oktatási és kutatási feladatok: Klincsik Mihály és Maróti György [193], Molnárka, Gergó, Wettl, Horváth, Kallós, [257] Szili László és Tóth János [356], valamint Stoyan Gisbert [345] könyvei.

Absztrakt adattípusok megvalósításával foglalkozik Horváth Zoltán, Kozsik

Tamás és Venczel Tibor cikke [145].

3. Algoritmusok (AL). Algoritmikai alap fogalmak, alapvető algoritmusok, osztott és párhuzamos algoritmusok, kriptográfiai algoritmusok, automataelmélet, geometriai algoritmusok, kiszámíthatóság, algoritmusok önálló elemzése).

A problémák *bonyolultság szerinti osztályozásával* kapcsolatban magyarul is hozzáférhető Ausiello [18] és Papadimitriou [282] könyve. Angolul Garey és Johnson klasszikus műve [106] mellett figyelmet érdemelnek Ausiello és társszerzői [19], Jones [174], Li és Vitanyi [237], Lovász [240], Sipser [338], Vogel és Wagner [378], valamint Wagner és Wechsung [379] eredményei. A *szóbonyolultság* számos kutatójának adatai megtalálhatók Iványi Antal honlapján [165].

A *közelítő algoritmusokkal* kapcsolatban gazdag anyagot tartalmaznak Hochbaum [133] és Vazirani [373] könyvei.

A *genetikus algoritmusokkal* foglalkozik magyarul Álmos Attila, Győri Sándor, Horváth Gábor és Várkonyiné Kóczy Annamária [7], angolul pedig Goldberg [113], Langdon és Poli [227], valamint Mitchell [256].

A *kriptográfiával* kapcsolatban Dénes József és Keedwell [65], valamint Salomaa, Rozenberg és Brauer [325] könyvét említjük.

A *szimulációval* foglalkozik Kátai Imre [181], valamint Aven, Coffman és Kogan [20].

Csirik János és Woeginger [60] összefoglalta a *közvetlen pakolási és lefedési algoritmusok* témakörét.

4. Programozási nyelvek (PL). Konkrét programozási nyelvek és rendszerezésük, absztrakt adattípusok, objektumelvű programozás, funkcionális programozás, fordítóprogramok.

A programozási nyelvek magyar nyelvű irodalma gazdag. A korai művek közül Farkas Zsuzsa, Futó Iván, Langer Tamás, és Szeredi Péter [77], Iványi Antalné és Kovács Zoltán [166], Jakobi Gyula [170], Horowitz [136], Kőhegyi János [218, 219, 220, 221], Kozics Sándor [215], Lőcs Gyula és Vigassy József [242, 243], Pirkó József [293, 292], Pongor György [294], Pyle [299], Sipos Annamária [337], Szlávi Péter és és Zsakó László [358, 393, 394] könyveit, az újak közül pedig Fóthi és Steingart [90], valamint Stroustrup [349] művét, továbbá a Nyékyné Gaizler Judit szerkesztésében megjelent könyveket [273, 274, 275] említjük.

Automatákkal és formális nyelvekkel foglalkozik magyarul Bach Iván [23] és Fülöp Zoltán [100] tankönyve, Csörnyei Zoltán [63], valamint Hunyadvári László és Manherz Tamás elektronikus jegyzete [149], Trahtenbrot [365] könyve, angolul pedig Hopcroft, Motwani és Ullman friss monográfiája [135].

5. Számítógépek felépítése (AR). Az alapok számábrázolás, funkcionális szervezés, multiprogramozás, hálózatok és osztott rendszerek felépítése.

Számítógépek felépítésével kapcsolatos Kovács Győző [211], Cserny László [56], Knuth [197], valamint Tanenbaum [360] könyve.

Az *elektronikus áramkörök tervezésének* algoritmusait foglalja össze Arató Péter, Jankovits István és Visegrády Tamás könyve [14].

Párhuzamos rendszerek *hardverének és szoftverének* kérdéseivel foglalkozik Amestoy [9], Hennessy és Patterson [130], Hwang [150], Kacsuk és Kotsis [177], Kacsuk, Kranzlmüller, Németh és Volkert [178].

6. Operációs rendszerek (OS). Konkrét rendszerek, elvek, párhuzamos folyamatok, ütemezés, memóriaszervezés, fájlszervezés, biztonság és védelem, hatékonyság.

Ajánlott irodalom: a *rendszerprogramozási algoritmusok* összefoglalása magyar nyelven megtalálható Csörnyei Zoltán [62], Donovan [69], Galambos Gábor [102], Tanenbaum és Woodhull [362], valamint Varga László [370, 371] könyveiben, idegen nyelven pedig Silberschatz, Galvin és Gagne [333], valamint Tanenbaum [361] műveiben.

Chow és Johnson [46], Malyshkin [246], valamint Tanenbaum és van Steen [363] könyvei az osztott rendszerek működésével foglalkoznak.

Norton és Stockman könyve [272] a *biztonsági kérdések* jó összefoglalója.

Az *ütemezési algoritmusok* irodalmából P. Brucker [38], French [93], Iványi Antal és Szmeljánszkij [168], Vizvári Béla [377] könyvét, valamint Iványi Antal elektronikus jegyzetét [162] említjük.

Petri-hálókról szól Kotov klasszikus [206], valamint Girault és Valk [111] most megjelent könyve.

7. Hálózatok (NC). Kommunikáció, biztonság, adattömörítés, konkrét webalkalmazások készítése, vezetékmentes számítások.

A *sejtautomatákkal* kapcsolatos cikkeket tartalmaz a Drommerné Takács Viola [72] által zerkezstett könyv. A *neurális hálókkal* kapcsolatos angol nyelvű irodalomból Chua és Roska Tamás [47], valamint Roska Tamás és Rodríguez-Vázquez [316] könyvét, továbbá Marczell Zsolt, Szepesváry Csaba, Kalmár Zsolt és Lőrincz András cikkét [247] emeljük ki.

Hálózatokkal foglalkozik magyarul Jutasi István [176], angolul Tanenbaum [359].

Idegen nyelven a *neurális hálókkal* kapcsolatban Chua és Roska Tamás [47] valamint Roska Tamás és Rodríguez-Vázquez [316] könyvét, továbbá Marczell Zsolt, Szepesváry Csaba, Kalmár Zsolt és Lőrincz András cikkét [247] emeljük ki.

8. Ember-számítógép kapcsolatok (HC).

9. Grafika (GV). Alapvető módszerek, grafikus kommunikáció, geometriai modellezés, animáció, láthatóság, virtuális valóság.

Ajánlott irodalom *számítógépi grafikáról* magyarul Füsi János [101] és Szirmay-Kalos László [357] könyve, valamint Vida János honlapja [374].

Geometriai modellezéssel foglalkozik de Berg, van Kreveld, Overmas és Schwarzkoff monográfiája [29] és Farin [76] könyve.

Az *animációval* kapcsolatban magyarul hozzáférhető Nagy Tibor [262] és Salamon Gábor [321] elektronikus gyűjteménye. Idegen nyelven nagy értéket képvisel Gloor, Dynes és Lee [112] CD-je, amelyen Cormen, Leiserson és Rivest [53] könyvének hiperszövege és animált algoritmusai is megtalálhatók. Parent friss könyve [286], Stasko honlapja [389], valamint Jürgen Winkler [387] elektronikus gyűjteménye is említésre méltó.

10. Intelligens rendszerek (IS). Tudás ábrázolása, következtetés, fejlett keresés, ágensek, természetes nyelv feldolgozása, gépi tanulás, robotika.

A *mesterséges intelligencia* számos algoritmusa megtalálható magyarul Fekete István, Gregorics Tibor és Nagy Sára [78], Futó Iván, Fekete István, Gregorics Tibor, Gyimóthy Tibor, Nagy Sára, Tatai Gábor és Sántáné Tóth Edit [99], Kelemen József és Nagy Sára [185], valamint Russel és Norvig [317] könyvében.

11. Információs rendszerek (IM). Adatmodellezés, lekérdező nyelvek, relációs adatbázisok, osztott adatbázisok, adatbányászás, digitális könyvtárak, hiperszöveg, multimédia.

Ajánlott irodalom magyarul *adatbázisokkal* kapcsolatban Garcia-Molina, Ullman és Widom [367, 105], valamint Rolland [313] könyvei. Az *adatbányászat* alapjait tartalmazza Adriaans és Zantinge műve [2]. A *szakértői rendszereket* elemzi Sántáné Tóth Edit jegyzete [326]. *Nagy adathalmazok* kezelésével foglalkozik Abello, Pardalos és Resende monográfiája [1].

12. Szoftvertechnológia (SE). Tervezés, eszközök és környezetek, folyamatok, követelmények és specifikáció, helyességbizonyítás, fejlesztés, csoportmunka irányítása, megbízhatóság.

A *feladatok specifikációjával* is foglalkozó magyar nyelvű anyagok közül ajánljuk például Sike Sándor és Varga László [372], valamint Sommerville [341] tankönyveit.

Párhuzamos programozásról szól Horváth Zoltán [144] és Szeberényi Imre [352] elektronikus kézirata, Kozma László és Varga László [216], valamint Chandy és Misra [43, 255] könyvei.

13. Társadalomtudományi alkalmazások (SP). Informatika története, társadalmi környezet, elemzés eszközei és módszerei, szakmai és erkölcsi felelősség, kockázatok, számítógépi jog, filozófia.

Ajánlott irodalom: magyarul Boros László [35] és Kurtán Lajos [222, 223] könyvei.

14. Kiszámíthatóság (CN). A fő témák numerikus módszerek, operációkutatás, nagy méretű számítások.

Numerikus módszereket ismertet magyarul Galántai Aurél és Jenei András [104], Gergó Lajos [108], Kiss Ottó és Kovács Margit [191], Henrici [131], Obádovics J. Gyula [276], Ralston [305], Simon Péter [336], Stoyan Gisbert és Takó Galina [346, 347, 348] és Szidarovszky Ferenc [353] könyve. Az angol nyelvű szakirodalomból Argyros, Bahill, Okuguchi, Szidarovszky Ferenc és Yakowitz [15, 354, 355], az orosz nyelvűből pedig N. N. Bahvalov, E. P. Zsidkov és G. M. Kobelkov [25] könyveit emeljük ki.

A numerikus módszerek alapját képező klasszikus analízis bizonyos részeit tárgyalja Járai Antal [171] könyve.

Az *optimalizálás módszereivel* kapcsolatos magyar nyelvű irodalomból Bajalinov Erik és Imreh Balázs [26], Imreh Balázs [152], Komlósi Sándor [203] könyvét, valamint Frank András [92] és Szántai Tamás elektronikus jegyzetét [351] említjük. Angol nyelvű Censor és Stavros [42] könyve, a Floudas és Pardalos által szerkesztett hétkötetes enciklopédia [83], valamint Pardalos és Resende [285] kézikönyve.

Operációkutatással foglalkozik Kovács Margit nyomtatott [212] és elektronikus [213], valamint Vizvári Béla [376] hagyományos jegyzete és Szántai Tamás [350] könyve.

A *játékelmélettel* kapcsolatban Morgenstern és Neumann János klasszikus műve [259] mellett Forgó Ferenc, Szép Jenő és Szidarovszky Ferenc [86], Kiss Béla és Krebsz Anna [190], valamint Okuguchi és Szidarovszky Ferenc könyvét [279] ajánljuk.

A *fuzzy rendszerek* elméletével kapcsolatos friss könyvek közül Carlsson és Fullér Róbert [41, 98] műveit említjük meg.

Fourier-analízissel foglalkoznak Weisz Ferenc [382, 383] könyvei. Schipp Ferenc, Wade, Simon Péter és Pál Jenő [327] monográfiájának témája a *harmonikus analízis*.

A valószínűségszámítás eszközeinek különböző alkalmazásairól szólnak Györfi Lászlónak és társszerzőinek a művei [66, 121, 122, 123].

Jelölések

A könyvben a következő jelöléseket alkalmazzuk.

$\mathcal{A}, \mathcal{B}$: soros algoritmusok

$b(n,k) = \binom{n}{k}$: n alatt a k binomiális együttható

$B(n,\ \pi,\ p,\ \mathcal{P})$: a $\mathcal{P}$ párhuzamos algoritmus legjobb lépésszáma

$B(n,\ \pi,\ \mathcal{A})$: az $\mathcal{A}$ soros algoritmus legjobb lépésszáma p processzoron

$C(n,\ \pi,\ p,\ \mathcal{P})$: a $\mathcal{P}$ párhuzamos algoritmus legrosszabb üzenetszáma

$g(n,\ \pi,\ \mathcal{A},\ \mathcal{P})$: a $\mathcal{P}$ párhuzamos algoritmus gyorsítása

$h(n,\ \pi,\ p,\ \mathcal{P})$: a $\mathcal{P}$: párhuzamos algoritmus hatékonysága

$\lg n = \log_2 n$: kettes alapú logaritmus

$\ln n = \log_e n$: természetes alapú logaritmus

$\lg^k n = (\lg n)^k$: logaritmus függvény hatványa

$\log n = \log_{10} n$: tízes alapú logaritmus

$\log^\star n$ = iterált logaritmus függvény

n: probléma mérete

$N(n, \pi, p, \mathcal{P})$: a $\mathcal{P}$ párhuzamos algoritmus szükséges lépésszáma p processzoron

$N(n, \pi, \mathcal{A})$: az $\mathcal{A}$ soros algoritmus szükséges lépésszáma

o: kis ordó

O': abszolút nagy ordó

$\tilde{O}$: gyenge nagy ordó

O: nagy ordó

$\overline{O}$: nagy valószínűségű nagy ordó

$\overset{\infty}{O}$: végtelen nagy ordó

p: processzorok száma

$\mathcal{P}$, $\mathcal{Q}$: párhuzamos algoritmusok

$W(n, \pi, p, \mathcal{P})$: a $\mathcal{P}$ párhuzamos algoritmus legrosszabb lépésszáma
$W(n, \pi, \mathcal{A})$: az $\mathcal{A}$ soros algoritmus legrosszabb lépésszáma

ω: kis omega

Ω: nagy omega

$\overline{\Omega}$: nagy valószínűségű nagy-omega

$\overset{\infty}{\Omega}$: végtelen nagy omega

π: probléma

Σ: véges ábécé

Θ: nagy teta

$\overline{\Theta}$: nagy valószínűségű nagy teta

$\lfloor\ \rfloor$: alsó egész rész

$\oplus$: bináris asszociatív operátor

$\times$: Descartes-szorzat

$:=$: értékadás jele

$\wedge$ és **and:** logikai **és**

!: faktoriális

$\lceil\ \rceil$: felső egész rész

$|\ |$: halmaz elemszáma

$\clubsuit$: treff (definíció jele)

$\blacksquare$: fekete kocka (bizonyítás végének a jele)

$\triangleright$: megjegyzés szimbólum

: : olyan, mint

$\spadesuit$: pikk (példa végének a jele)

$\vee$ és **or:** logikai **vagy**

Angol kifejezések

Ebben a részben megadjuk a témakör angol nyelvű irodalmából átvett szakkifejezések magyar megfelelőjét.

anomaly = anomália

average degree = átlagos fokszám

average level = átlagos szint

big oh = nagy ordó

big omega = nagy omega

big theta = nagy teta

binary tree network = bináris fa hálózat

butterfly = pillangó

concurrent read, concurrent write (CRCW) = párhuzamos olvasás, párhuzamos írás

concurrent read, exclusive write (CREW) = párhuzamos olvasás, kizárólagos írás

cross link = kereszt kapcsolat

de Bruijn network = de Bruijn-hálózat

degree = fokszám

direct link = közvetlen kapcsolat

efficiency = hatékonyság

exclusive read, exclusive write (EREW) = kizárólagos olvasás, kizárólagos írás

exclusive read, concurrent write (ERCW) = kizárólagos olvasás, párhuzamos írás

farthest destination first (FDF) = legtávolabbra utazó csomag először

farthest origin first (FOF) = legtávolabbról jött csomag először

first in first out (FIFO) = előbb be, előbb ki

Hamming distance = Hamming-távolság

hipercube = hiperkocka

LCCB (least cost branch-and-bound) = legkisebb költségű korlátozás és szétválasztás

linear speedup = lineáris gyorsítás

linear running time = lineáris lépésszám

maximal degree = maximális fokszám

minimal degree = minimális fokszám

mesh = rács

most significant bits (MSB) = legnagyobb helyi értékű bitek

online algorithm = közvetlen algoritmus

packet routing (PR) = csomagirányítás

partial permutation routing (PPR) = parciális permutáció irányítás

parallel hypercube = párhuzamos hiperkocka

parallel random access machine (PRAM) = párhuzamos közvetlen hozzáférésű gép

planar graph = síkgráf

planar net = síkhálózat

pyramid network = piramis hálózat

random access machine (RAM) = közvetlen hozzáférésű gép

sequential hypercube = soros hiperkocka

speedup = relatív sebesség

stack property = verem tulajdonság

star network = csillag hálózat

sublinear speedup = szublineáris gyorsítás

sublogarithmic running time = szublogaritmikus futási idő

superlinear speedup = szuperlineáris gyorsítás

total work = összes munka

work-optimal parallel algorithm = munkahatékony párhuzamos algoritmus

Magyar szakkifejezések

Ebben a részben összefoglaljuk a könyvben használt magyar szakkifejezések angol megfelelőit.

anomália = anomaly
átlagos fokszám = average degree
átlagos szint = average level
bináris fa hálózat = binary tree network
csillag hálózat = star network
csomagirányítás = packet routing (PR)
de Bruijn-hálózat = de Bruijn network
előbb be, előbb ki = first in first out (FIFO)
fokszám = degree
Hamming-távolság = Hamming distance
hatékonyság = efficiency
hiperkocka = hipercube
kereszt kapcsolat = cross link
kizárólagos olvasás, kizárólagos írás = exclusive read, exclusive write (EREW)
kizárólagos olvasás, párhuzamos írás = exclusive read, concurrent write (ERCW)
közvetlen algoritmus = online algorithm

közvetlen hozzáférésű gép = random access machine (RAM)

közvetlen kapcsolat = direct link

legkisebb költségű korlátozás és szétválasztás = LCCB (least cost branch-and-bound) =

legnagyobb helyi értékű bitek = most significant bits (MSB)

legtávolabbra utazó csomag először = farthest destination first (FDF)

legtávolabbról jött csomag először = farthest origin first (FOF)

lineáris futási idő = linear running time

lineáris gyorsítás = linear speedup

maximális fokszám = maximal degree

minimális fokszám = minimal degree

munkahatékony párhuzamos algoritmus = work-optimal parallel algorithm

nagy o = big oh

nagy omega = big omega

nagy teta = big theta

összes munka = total work

parciális permutáció irányítás = partial permutation routing (PPR)

párhuzamos hiperkocka = parallel hypercube

párhuzamos közvetlen hozzáférésű gép = parallel random access machine (PRAM)

párhuzamos olvasás, kizárólagos írás = concurrent read, exclusive write (CREW)

párhuzamos olvasás, párhuzamos írás = concurrent read, concurrent write (CRCW)

pillangó = butterfly

piramis hálózat = pyramid network

rács = mesh

relatív sebesség = speedup

síkgráf = planar graph

síkhálózat = planar net

soros hiperkocka = sequential hypercube

szublineáris gyorsítás = sublinear speedup

szublogaritmikus futási idő = sublogarithmic running time

szuperlineáris gyorsítás = superlinear speedup

verem tulajdonság = stack property

Irodalomjegyzék

[1] J. Abello, P. M. Pardalos, M. G. C. Resende: *Handbook of Massive Data Sets*. Kluwer Academic Publishers, Dordrecht, 2002. 1236 oldal. ISBN 1-4020-0489-3. 6.5

[2] P. Adriaans, D. Zantinge: *Data Mining*. Addison-Wesley Longman, 1996. Magyarul: Adatbányászat. Panem, Budapest, 2002. 157 oldal, 39 hivatkozás. ISBN 963 545 367 1. 6.5

[3] A. V. Aho, J. E. Hopcroft, J. D. Ullman: *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, 1974. 240 oldal, 235 hivatkozás, ISBN 0-201000-296. Magyarul: *Számítógép-algoritmusok tervezése és analízise.* Műszaki Könyvkiadó, Budapest, 1982. 487 oldal, 235 hivatkozás, ISBN 963 104 323 1. 6.1

[4] A. V. Aho, J. E. Hopcroft, J. D. Ullman: *Data Structures and Algorithms*. Addison-Wesley, Reading, 1983. 427 oldal, 158 hivatkozás, ISBN 0-201-00023-7. 6.1

[5] S. G. Akl: *The Design and Analysis of Parallel Algorithms*. Prentice-Hall, Englewood Cliffs, 1989. 400 oldal, ISBN 0132000563. 6.1

[6] G. S. Almasi, A. Gottlieb: *Highly Parallel Computing*. The Benjamin/Cummings Publ. Comp., Redwood City, 1994. 669 oldal, 472 hivatkozás, ISBN 0-8053-0443-6. 6.1

[7] Álmos Attila, Győri Sándor, Horváth Gábor, Várkonyiné Kóczy Annamária: *Genetikus algoritmusok*. Typotex, Budapest,2002. 255 oldal, 137 hivatkozás. 6.5

[8] G. M. Amdahl: Validity of the single-processor approach to achieving large-scale computer capabilities. In: *AFIPS Conference Proceedings* **30** 1967, 483–485. 6.1

[9] P. Amestoy (szerkesztő): *Euro-Par'99*. *LNCS* **1685**. Springer-Verlag, Berlin, 1999, 1503 oldal. ISBN 3540 664432 6.1, 6.5

[10] Andrásfai Béla: *Ismerkedés a gráfelmélettel*. Második kiadás. Tankönyvkiadó, Budapest, 1973. 237 oldal, 48 hivatkozás, ISBN 363 178 663 3. 6.2

[11] Andrásfai Béla: *Gráfok. Mátrixok és folyamok*. Akadémiai Kiadó, Budapest, 1983. 263 oldal, 74 hivatkozás, ISBN 963 05 31461. 6.2

[12] Andrásfai Béla: *Gráfelmélet*. Polygon Kiadó, Szeged, 1994. 174 oldal, 10 hivatkozás, ISBN 1417-0590. 6.2

[13] Andrásfai Béla: *Infor-matek*. Polygon Kiadó, Szeged, 1997. 282 oldal, 13 hivatkozás, ISBN 1218-4071. 6.1, 6.5

[14] Arató Péter, Jankovits István, Visegrády Tamás: *High-Level Synthesis of Pipelined Datapaths*. Panem/John Wiley & Sons, Budapest, 1999. 251 oldal, ISBN 0471495824. 6.5

[15] I. K. Argyros, F. Szidarovszky: *The Theory and Applications of Iteration Methods*. CRC Press Inc., Boca Raton, 1993. 335 oldal, 110 hivatkozás, ISBN 0-8493-8014-6. 6.5

[16] L. Artiaga, L. Davis: *Algoritmusok és FORTRAN programjaik*. Műszaki Könyvkiadó, Budapest, 1977. 6.1

[17] M. J. Atallah: *Algorithms and Theory of Computation Handbook*. CRC Press, Boca Raton, 1999. 1218 oldal, 4118 hivatkozás, ISBN 0-8493-2649-4. 6.1

[18] G. Ausiello: *Complessitá di calcolo delle functioni*. Bodoglierie Societá, Torino, 1975. Magyarul: Algoritmusok és rekurzív függvények bonyolultságelmélete. Műszaki Könyvkiadó, Budapest, 1984. 67 oldal, 109 hivatkozás, ISBN 963 10 5159 5. 6.5

[19] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, M. Protasi: *Complexity and Approximation*. Springer-Verlag, Berlin, 1999. 524 oldal, 602 hivatkozás, ISBN 3504-65431-3. 6.5

[20] O. I. Aven, E. G. Coffman jr., Y. A. Kogan: *Stochastic Analysis of Computer Storage*. D. Reidel Publ. Company, Dordrecht, 1987. 254 oldal, 157 hivatkozás, ISBN 90-277-2515-2. 6.5

[21] S. Baase: *Computer Algorithms: Introduction to Design and Analysis*. Második kiadás. Addison-Wesley, 1988. 435 oldal, 131 hivatkozás. ISBN 0-201-06035-3. 6.1, 6.5

[22] S. Baase, A. Van Gelder: *Computer Algorithms: Introduction to Design and Analysis*. Harmadik kiadás. Addison-Wesley, Reading, 2000. 688 oldal, ISBN 0-201-61244-5. 6.1, 6.5

[23] Bach Iván: *Formális nyelvek*. Typotex, Budapest, 2001. 6.5

[24] Bagyinszki János, György Anna: *Diszkrét matematika főiskolásoknak*. Typotex, Budapest, 2001. 152 oldal, ISBN 9639132969. 6.5

[25] N. N. Bahvalov, E. P. Zhidkov, G. M. Kobelkov *Numerical Methods*. Fizmatlit, Moscow, 2000. 624 oldal. 6.5

[26] Bajalinov Erik, Imreh Balázs: *Operációkutatás*. POLYGON Kiadó, Szeged, 2001. 6.5

[27] G. H. Barnes: The ILLIAC-IV computer. *IEEE Transactions on Computers* **C-17** 1968, 746–757. 6.1

[28] Belényesi Viktor, Németh Cs. Dávid: *d*-bonyolultság számítása. TDK-dolgozat. ELTE, Informatikai Tanszékcsoport, Budapest, 2002. 17 oldal. 6.1

[29] M. de Berg, M. van Kreveld, M. Overmas, O. Schwarzkoff: *Computational Geometry*. Második kiadás. Springer-Verlag, Berlin, 2000. 367 oldal, 348 hivatkozás, ISBN 3-540-65620-0. 6.5

[30] C. Berge: *Graphs and Hypergraphs*. North-Holland Publishing Company, Amsterdam, 1976. 528 oldal, 542 hivatkozás, ISBN 0 7204 2453 4. 6.2

[31] Berke Barnabásné: *ISBN útmutató*. Második kiadás. Országos Széchenyi Könyvtár, Budapest, 1991. 22 oldal, ISBN 963 200 305 5. 394

[32] K. A. Berman, J. L. Paul: *Fundamentals of Sequential and Parallel Algorithms*. PWS Publishing Company, Boston, 1997. XXIII+744 oldal, 80 hivatkozás, 0-534-94674-7. 6.1, 6.2, 6.3

[33] H. L. Bodlaender: A better lower bound for distributed leader finding in bidirectional asynchronous rings of processors. *Information Processing Letters* **27** 1988, 287–290. 6.5

[34] B. Bollobás: *Random Graphs*. Academic Press, London, 1985. XII+441 oldal, 757 hivatkozás. ISBN 0-12-11756-1. 6.1, 6.2

[35] Boros, László: *Jogi alapismeretek*. Vince Kiadó, Budapest, 1998. 154 oldal. 6.5

[36] G. Brassard, P. Bratley: *Fundamentals of Algorithms*. Prentice Hall, Englewood Cliffs, 1996. 524 oldal, 355 hivatkozás, ISBN 0-13-335068-1. 6.1

[37] I. N. Bronstejn, I. N. Szemengyajev et al.: *Matematikai kézikönyv*. 8. átdolgozott kiadás. Typotex, Budapest, 2002. 1209 oldal, ISBN, 9639132594. 6.5

[38] P. Brucker: *Scheduling Algorithms*. Springer-Verlag, Berlin, 1998. 342 oldal, 216 hivatkozás, ISBN 3-540-64105-X. 6.5

[39] R. G. Busacker, T. L. Saaty: *Finite Graphs and Networks.* McGraw Hill and Company, New York, 1965. XIV+294 oldal, hivatkozás. Magyarul: *Véges gráfok és alkalmazásaik*. Műszaki Könyvkiadó, Budapest, 1969. 347 oldal. 6.2

[40] CALGO (*Collected Algorithms of ACM*). Association for Computing Machinery, 2003. 6.1

[41] C. Carlsson, R. Fullér: *Fuzzy Reasoning in Decision Making and Optimization*. *Studies in Fuzziness and Soft Computing Series* **82**. Springer-Verlag, Berlin/Heildelberg, 2002. 338 oldal, ISBN 3-7908-1428-8. 6.5

[42] Y. Censor, A. Z. Stavros: *Parallel Optimization: Theory, Algorithms, and Applications*. Oxford University Press, New York, 1997, XVII+539 oldal, 566 hivatkozás. ISBN 0-19-510062–X. 6.5

[43] K. M. Chandy, J. Misra: *Parallel Program Design: A Foundation*. Addison-Wesley, 1988. 6.5

[44] E. J.-H. Chang, R. Roberts: An improved algorithm for decentralized extrema finding in circular arrangement of processes. *Communication of ACM* **22** 1979, 281–283. 6.5

[45] H. Chernoff: A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics*, **23** 1952, 493–507. 6.1

[46] R. Chow, T. Johnson: *Distributed Operating Systems and Algorithms*. Addison-Wesley, Reading, 1997. 569 oldal, 512 hivatkozás, 0-201-49838-3. 6.5

[47] L. Chua, T. Roska: *Cellular Neural Networks and Visual Computing - Foundations and Applications*. Cambridge University Press, Cambridge, 2001. 380 oldal, ISBN 0521652472. 6.5

[48] E. G. Coffman, Jr.: *Computer and Job Shop Scheduling*. John Wiley & Sons, New York, 1976. 299 oldal, 158 hivatkozás, ISBN 0-471-16319-8. 6.1

[49] E. G. Coffman jr., J. Csirik, G. J. Woeginger: *Approximate solutions to bin packing problems*. In: [285].

[50] E. G. Coffman, G. Galambos, S. Martello, D. Vigo: Bin packing approximation algorithms: combinatorial analysis. In [73], 151–208. 6.1

[51] R. Cole, U. Vishkin: Deterministic coin-tossing with applications to optimal parallel list ranking. *Information and Control* **70** 1986, 32–53.

[52] T. H. Cormen, C. Lee, E. Lin: *Instructor's Manual*. The MIT Press, Cambridge, 2002. 402 oldal. 6.1

[53] T. H. Cormen, C. E. Leiserson, R. L. Rivest: *Introduction to Algorithms*. The MIT Press/McGraw-Hill, Cambridge/New York, 1990, 1028 oldal, 205 hivatkozás. ISBN 0-262-03141-8. Magyarul: *Algoritmusok*. Harmadik kiadás. Műszaki Könyvkiadó, Budapest, 2001. XVI+884 oldal, 205+65 hivatkozás, ISBN 963 163029 3. 6.1, 6.4, 6.5

[54] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein: *Introduction to Algorithms*. The MIT Press/McGraw-Hill, Cambridge/New York, 2001. XI+1180 oldal, 320 hivatkozás. ISBN 0-262-03293-7. Magyarul: *Új algoritmusok*. Scolar, Budapest, 2003 szeptember (előkészületben). 1020 oldal, ISBN 963 9193 90 9. 6.1

[55] Cseke Vilmos: *A gráfelmélet és alkalmazásai*. Tudományos Könyvkiadó, Bukarest, 1972. 165 oldal, 8 hivatkozás. 6.2

[56] Cserny László: *RISC processzorok*. LSI, Budapest, 1996. 403 oldal, 34 hivatkozás, ISBN 963 577 155 X. 6.5

[57] J. Csirik, G. Galambos, J. B. G. Frenk, A. H. G. Rinnooy Kan: A probabilistic analysis of a dual bin packing problem. *Journal of* Algorithms **12** (1987), 189-203. 6.1

[58] J. Csirik, B. Imreh: On the worst case behaviour of the Next-k Fit bin packing algorithm. *Acta Cybernetica* 1989, 89–95. 6.1

[59] J. Csirik, D. S. Johnson: Best is better than first. *Algoritmica*. **31** 2001, 115–138. 31 hivatkozás. 6.1

[60] J. Csirik, G. J. Woeginger: On-line packing and covering problems. In: [81], 147–177. 6.5

[61] Csizmazia Balázs: *Hálózati alkalmazások készítése*. Kalibán Bt., Budapest, 1998. 362 oldal, 16 hivatkozás, ISBN 963 04 9630 5.

[62] Csörnyei Zoltán: *Fordítási algoritmusok*. Erdélyi Tankönyvtanács, Kolozsvár, 2000. VII+195 oldal, 13 hivatkozás, ISBN 973 99814 8 8. 6.5

[63] Csörnyei Zoltán: *Típuselmélet (Kombinátor logika. λ-kalkulus. Típusos λ-kalkulus*. Elektronikus jegyzet. ELTE Informatikai Tanszékcsoport, Budapest, 2003. 6.5

[64] Demetrovics János, Denev J., Pavlov R.: *Bevezetés a számítástudományba*. Tankönyvkiadó, Budapest, 1982. 374 oldal, 26 hivatkozás, ISBN 963 118463 8 6.1, 6.5

[65] J. Dénes, A. D. Keedwell: *Latin Squares*. North-Holland, Amsterdam, 1991. XIV+453 oldal, ISBN 0-444-88899-3. 6.5

[66] L. Devroye, L. Györfi, G. Lugosi: *A Probabilistic Theory of Pattern Recognition*. Springer-Verlag, New York, 1996. 6.5

[67] R. Diestel: *Graph Theory*. Springer-Verlag, Berlin, 1997. 312 oldal, ISBN 0-387-95014-1. 6.2

[68] D. Dolev, M. Klawe, M. Rodeh: An $O(N \log N)$ unidirectional distributed algorithm for extrema-finding in circle. *Journal of* Algorithms **3** 1982, 245-260. 6.5

[69] J. J. Donovan: *Systems Programming*. McGraw-Hill, 1972. Magyarul: *Rendszerprogramozás*. Műszaki Könyvkiadó, Budapest, 1976. 6.5

[70] J. A. Dossey, A. D. Otto, L. E. Spence, C. W. Eynden: *Discrete Mathematics*. Scott, Foresman and Company, Glenview, 1972. 482 oldal, 45 hivatkozás, ISBN 0-673-18191-X. 6.5

[71] Dringó László, Kátai Imre: *Bevezetés a matematikába*. Tankönyvkiadó, Budapest, 1977. 288 oldal. 6.1, 6.5

[72] Drommerné Takács Viola (szerkesztő): *Sejtautomaták*. Gondolat, Budapest, 1978. 288 oldal, 176 hivatkozás, ISBN 963 280 665 4. 6.1, 6.5

[73] D.-Z. Du, P. M. Pardalos: *Handbook of Combinatorial Optimization. Supplement Volume*. Kluwer Academic Publishers, 1999. 656 oldal, ISBN 0-7923-5924-0. 50

[74] P. Erdős: *The Art of Counting* (szerkesztő J. Spencer). The MIT Press, Cambridge, 1973. 742 oldal, 596 hivatkozás, ISBN 0-262-19116-4. 6.5

[75] P. Erdős, J. Spencer: *Probabilistic Methods in Combinatorics*. Akadémiai Kiadó/Academic Press, Budapest/New York, 1974. 106 oldal, 119 hivatkozás. 6.1

[76] G. E. Farin: *Curves and Surfaces for Computer Aided Geometric Design. A Practical Guide*. Negyedik kiadás. Academic Press, 1998. 6.5, 374

[77] Farkas Zsuzsa, Futó Iván, Langer Tamás, Szeredi Péter: *MPROLOG programozási nyelv*. Műszaki Könyvkiadó, Budapest, 1986. 280 oldal, 8 hivatkozás. 6.5

[78] Fekete István, Gregorics Tibor, Nagy Sára: *Bevezetés a mesterséges intelligenciába*. LSI Kiadó, Budapest, 1990. 289 oldal, 28 hivatkozás. Gábor Dénes Főiskola, Budapest. 1999. 292 oldal, 33 hivatkozás. 6.5

[79] W. Feller: *An Introduction to Probability Theory and its Applications*. Harmadik kiadás. John Wiley & Sons, New York, 1968. Magyarul: *Bevezetés a valószínűségszámításba*. Műszaki Könyvkiadó, Budapest, 1978. 478 oldal, hivatkozás, ISBN 963 102 070 3. 6.1

[80] S. Ferenczi, Z. Kása: Complexity of finite and infinite words. *Theoretical Computer Science* **218** 1999, 177–195. MR2000d:68121, ZB916.68112 6.1

[81] A. Fiat, G. J. Woeginger: *Online Algorithms. The State of Art*. *LNCS* **1442.** Springer-Verlag, Berlin, 1998. 436 oldal, 1285 hivatkozás, ISBN 3-540-64917-4. 60

[82] P. Flach: *Simply Logical. Intelligent Reasoning by Example*. John Wiley Şons. 1989., London. 236 oldal, 45 hivatkozás. Magyarul: *Logikai progra-*

mozás. Panem, Budapest, 2001. 288 oldal, 45 hivatkozás, ISBN 963 545 297 7. 6.5

[83] C. A. Floudas, P. M. Pardalos: *Encyclopedia of Optimization* (7 kötet). Kluwer Academic Publishers, 2001, 3200 oldal, ISBN 0-7923-6932-7. 6.5

[84] R. W. Floyd, R. L. Rivest: Expected time bounds for selection *Communication of ACM* **18** (3) 1975, 165–172. 6.2

[85] M. J. Flynn: Very high-speed computer systems. *Proceedings of the IEEE* **5** (6) 1966, 1901–1909. 6.1

[86] F. Forgó, J. Szép, F. Szidarovszky: *Introduction to Games: Concepts, Methods and Applications*. Kluwer Academic Press, Boston, 1999. 352 oldal, 152 hivatkozás, ISBN 0-7923-5775-2. 6.5

[87] Fornai Péter, Iványi, Antal: Bélády's anomaly is unbounded. In: [209], 65–72. 13 hivatkozás. 6.1

[88] S. Fortune, J. Wyllie: Parallelissm in RAM's. In: *Proceedings of the 10-th ACM STOC*. 1978, 114-118.

[89] Fóthi Ákos: *Bevezetés a programozásba*. Tankönyvkiadó, Budapest, 1983. 101 oldal, 6.5

[90] Fóthi Ákos, Steingart Ferenc: *Bevezetés a programozáshoz*. Elektronikus kézirat. Budapest, ELTE Informatikai Tanszékcsoport, 2003. 130 oldal. 6.5

[91] Á. Fóthi, Z. Horváth, T. Kozsik: Parallel elementwise processing - a Novel version. *Annales Univ. Sci. Budapest de R. Eötvös Nom., Sectio Computatorica* **17** 1998, 105–124. MR 2000c:68150, ZB 0981.68177.

[92] Frank András: *Kombinatorikus algoritmusok*. Elektronikus jegyzet. ELTE, Operációkutatási Tanszék, Budapest, 2003. 57 oldal. 6.5

[93] S. French: *Sequencing and Scheduling*. Ellis Horwood Limited, New York, 1982. 245 oldal, 234 hivatkozás, ISBN 0-85312-364-0. 6.5

[94] Freud Róbert: *Lineáris algebra*. ELTE Eötvös Kiadó, Budapest, 1996. 518 oldal, ISBN 963 463 080 4. 6.5

[95] Freud Róbert, Gyarmati Edit: *Számelmélet*. Nemzeti Tankönyvkiadó, Budapest, 2000. 740 oldal, ISBN 9631907848. 6.5

[96] Frey Tamás, Szelezsán János: *Matematikai kibernetika*. Akadémiai Kiadó, Budapest, 1973. 120 oldal. 6.1

[97] Frey Tamás, Szelezsán János: *Számítástechnika*. Akadémiai Kiadó, Budapest, 1973. 168 oldal. 6.1

[98] R. Fullér: *Introduction to Neuro-Fuzzy Systems*. *Advances in Soft Computing Series*. Springer-Verlag, Berlin/Heidelberg, 2000. 289 oldal, 46 hivatkozás, ISBN 3-7908-1256-0. 6.5

[99] Futó Iván, Fekete István, Gregorics Tibor, Gyimóthy Tibor, Nagy Sára, Sántáné Tóth Edit, Tatai Gábor. *Mesterséges intelligencia*. Aula Kiadó, Budapest, 1999. XIX+986 oldal, ISBN 963 907 899 9. 6.5

[100] Fülöp Zoltán: *Formális nyelvek és szintaktikus elemzésük*. Polygon, Szeged, 1999. 6.5

[101] Füsi János: *3D grafikai animáció*. LSI Kiadó, Budapest, 1993. 227 oldal, 7 hivatkozás, ISBN 963 618 111 X. 6.5

[102] Galambos Gábor: *Operációs rendszerek*. Műszaki Könyvkiadó, Budapest, 2003. ISBN 963 162284 3. 6.5

[103] Galántai Aurél, Hujter Mihály: *Optimalizálási módszerek*. Miskolci Egyetemi Kiadó, Miskolc, 1997.

[104] Galántai Aurél, Jenei András: *Numerikus módszerek*. Miskolci Egyetem Kiadó, Miskolc, 1998. 171 oldal, ISBN 963 661 3117. 6.5

[105] H. Garcia-Molina, J. D. Ullman, J. Widom: *Database System Implementation*. Prentice Hall, 2000. 654 oldal, 109 hivatkozás. Magyarul: *Adatbázisrendszerek megvalósítása* (szerkesztette Benczúr András). Panem, Budapest, 2001. 684 oldal, 109 hivatkozás, ISBN 9635452 80 2. 6.5

[106] M. R. Garey, D. S. Johnson: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco, 1979. 338 oldal, 763 hivatkozás. ISBN 0-7167-1044-7. 6.1, 6.5

[107] G. P. Gavrilov, A. A. Szapozsenko: *Diszkrét matematikai feladatgyűjtemény* (oroszul). Moszkva, Nauka, 1977, 367 oldal. Magyarul: *Diszkrét matematikai feladatgyűjtemény* (fordította Bagyinszki János). Műszaki Könyvkiadó, Budapest, 1981. 357 oldal, 36 hivatkozás, ISBN 963 103 779 9. 6.5

[108] Gergó Lajos: *Numerikus módszerek. Példák és feladatok*. ELTE Eötvös Kiadó, Budapest, 2000. 204 oldal, 16 hivatkozás. 6.5

[109] A. Gibbons, W. Rytter: *Efficient Parallel Algorithms*. Cambridge University Press, 1988, 259 oldal, 154 hivatkozás. ISBN 0 52134585 4. 6.1

[110] B. Gilchrist (szerkesztő): *Proceedings of Information Processing'77*. North-Holland, 1977. 235

[111] C. Girault, R. Valk (szerkesztők): *Petri Nets for System Engineering. A Guide to Modelling, Verification, and Applications*. Springer-Verlag, Berlin, 2003. 597 oldal, ISBN 3-540-41217-4. 6.5

[112] P. Gloor, C. Dynes, L. Lee: *Animated Algorithms. A Hypermedia Learning Environment for Introduction to Algorithms.* CD ROM. The MIT Press, 1993. 6.5

[113] D. E. Goldberg: *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publ. Comp. 1989. 412 oldal, ISBN 0201157675. 6.5

[114] J. Gruska: *Foundations of Computing*. International Thomson Computer Press, London, 1997. 716 oldal, 427 hivatkozás, ISBN1-85032-243-0. 6.1

[115] Gonda János: *Bevezetés a matematikába. III.*. ELTE TTK, Budapest, 1998. 295 oldal, 17 hivatkozás. 6.1, 6.5

[116] G. H. Gonnet, R. Baeza-Yates: *Handbook of Algorithms and Data Structures. In Pascal and C*. Addison-Wesley, Wokingham, 1991. 424 oldal, 1350 hivatkozás, ISBN 0-201-41607-7. 6.1

[117] D. H. Greene, D. E. Knuth: *Mathematics for the Analysis of Algorithms*. Birkhäuser, Boston, 1990. 132 oldal, 44 hivatkozás, ISBN 0-8176-3515-7. 6.1

[118] R. Greenflaw, J. Hoover, J. W. Ruzzo: *Limits to Parallel Computation: P-completeness Theory*. Oxford University Press, New York, 1995. ISBN 0 19 508591 4. 6.1

[119] J. Gruska: *Foundations of Computing*. International Thompson Publishing, London, 1997. XVII+716 oldal, 320 hivatkozás, ISBN 1-85032-243-0. 6.1

[120] J. Gustafson: Reevaluating Amdahl's law. *Communications of ACM* **28** (1) 1988, 532–535. 6.1

[121] L. Györfi, M. Kohler, A. Krzyzak, H. Walk: *A Distribution-Free Theory of Nonparametric Regression*. Springer-Verlag, New York, 2002. 6.5

[122] Györfi László, Győri Sándor, Vajda István: *Információ- és kódelmélet*. Typotex Kiadó, Budapest, 2000. 371 oldal, 42 hivatkozás, ISBN 963 9132 845. 6.5

[123] L. Györfi (szerkesztő) *Principles of Nonparametric Learning*. Springer-Verlag, Wien/New York, 2002. 6.5

[124] P. R. Halmos: *Naive Set Theory*. Springer-Verlag, New York, 1987. 104 oldal, 19 hivatkozás, ISBN 0387900926. Magyarul: *Elemi halmazelmélet* (fordította Kósa András). Műszaki Könyvkiadó, Budapest, 1981. 100 oldal, 19 hivatkozás, ISBN 963 10 3857 2. 6.1

[125] T. J. Harris: A survey of PRAM simulation techniques. *ACM Computing Surveys* **26** 1964, 164–206.

[126] Hatvani László, Imreh Balázs: *Kombinatorikus optimalizálás*. Novadat Bt., Győr, 1999.

[127] R. L. Haupt, S. E. Haupt: *Practical Genetic Algorithms*. Wiley, John & Sons, 1997. 192 oldaL, ISBN 0471188735.

[128] M. T. Heath, A. Ranade, R. S. Schreiber (szerkesztők): *Algorithms für Parallel Processors*. *IMA* **105.** Springer-Verlag, Berlin, 1999, 366 oldal. ISBN 0387-98680-4. 6.1

[129] H.-D. Hecker: *Übungsserie für Algorithmen und Datenstrukturen*. Friedrich Schiller Universität, Jena, 2003. 6.1

[130] J. L. Hennessy, D. A. Patterson: *Computer Architecture. A Quantitative Approach*. Morgan Kaufmann Publishers, San Mateo, 2002. 1136 oldal, ISBN 55860 596 7. 6.5

[131] R. Henrici: *Numerikus módszerek*. Műszaki Könyvkiadó, Budapest, 1985. 370 oldal, 56 hivatkozás, ISBN 963 10 6419 0. 6.5

[132] D. S. Hirschberg, J. B. Sinclair: Decentralized extrema-finding in circular configurations of processes. *Communications of* ACM **23** 1980, 627–628. 6.5

[133] D. S. Hochbaum: *Approximation Algorithms for NP-hard Problems*. PWS Publishing Company, Boston, 1995. 596 oldal, 683 hivatkozás, ISBN 0-534-94968-1. 6.5

[134] M. Hofri: *Probabilistic Analysis of Algorithms*. Springer-Verlag, New York, 1987. 240 oldal, 102 hivatkozás, ISBN 0-387-96578-5. 6.1

[135] J. E. Hopcroft, R. Motwani, J. D. Ullman: *Introduction to Automata Theory. Languages and Computation*. Addison-Wesley, Boston, 2001. 521 oldal, ISBN 0-201-44124-1. 6.5

[136] E. Horowitz: *Fundamentals of Programming Languages*. 2. kiadás. Computer Science Press, Rockville, 1984. Magyarul: *Magasszintű programozási nyelvek*. Műszaki Könyvkiadó, Budapest, 1987. 334 oldal, 236 hivatkozás, ISBN 963-10-7211-8. 6.5

[137] E. Horowitz, S. Sahni, S. Rajasekaran: *Computer Algorithms*. Computer Science Press, New York, 1998. 769 oldal. ISBN 0-7167-8316-9. 6.1, 6.2, 6.3, 6.4

[138] Horváth László, Pirkó József: *Számítástechnikai lexikon. 1. kötet. Az alapok*. Kossuth Könyvkiadó, Budapest, 1996. 227 oldal, ISBN 963 093 871 5. 6.1

[139] Horváth László, Pirkó József: *Számítástechnikai lexikon. 2. kötet. Windows*. Kossuth Könyvkiadó, Budapest, 1997. 138 oldal, ISBN 963 09 3872 3. 6.1

[140] Horváth László, Pirkó József: *Informatikai tudástár*. Kiskapu Kiadó, Budapest, 2001. 608 oldal, 12 hivatkozás, ISBN 9639301280. 6.1

[141] Horváth Márk, Iványi Antal: Construction of perfect right prisms. Technical Reports in Informatics of Eötvös Loránd University **2003-A03**. Budapest, 2003, 8 oldal. 6.1

[142] Horváth Zoltán: Parallel asynchronous computation of the values of an associative function. *Acta Cybernetica* **12** (1) (1995), 83-94. ZB0830.68084. 6.2

[143] Horváth Zoltán: The formal specification of a problem solved by a parallel program – a relational modell. *Annales Univ. Sci. Budapest., Sectio Computatorica* **18** 1998, 173–191. MR 2000g:68095.

[144] Horváth Zoltán: *Párhuzamos programozás alapjai*. Kézirat. ELTE Informatikai Tanszékcsoport, Budapest, 1996. 146 oldal. 6.2, 6.5

[145] Horváth Zoltán, Kozsik Tamás, Venczel Tibor: Parallel programs implementing abstract data type operations. *Pure Mathematics and Applications* **11** (2) (2000), 293-308. ZBpre01668902. 6.5

[146] Horváth Zoltán, Hernyák Zoltán, Kozsik Tamás, Tejfel Máté, Ulbert Attila: A data intensive computation on a cluster. In: [178], 49–53.

[147] Hunyadvári László: *Bevezetés a számítástudományba*. Elektronikus feladatgyűjtemény. ELTE Informatikai Tanszékcsoport, Budapest, 2003. 19 oldal.

[148] Hunyadvári László, Fekete István: *Algoritmusok és adatszerkezetek. Feladatgyűjtemény* a gyakorlatokhoz. Elektronikus kézirat. ELTE Informatikai Tanszékcsoport, Budapest. 2003. 13 oldal. 6.1

[149] Hunyadvári László, Manherz Tamás: *Formális nyelvek*. Elektronikus kézirat. ELTE, Informatikai Tanszékcsoport, 2003. 6.5

[150] K. Hwang: *Advanced Computer Architecture*. McGraw-Hill, Inc., New York, 1993. 672 oldal, ISBN 0 07 031622 8. 6.5

[151] IEEE, ACM: *Computing Curricula* 2001. Institut of Electrical and Electronics Engineering and Association for Computing Machinery, 2001. 220 oldal. 6.5

[152] Imreh Balázs: *Kombinatorikus optimalizálás*. Novadat, Győr, 1999. 208 oldal, 98 hivatkozás. 6.5

[153] Iványi Anna: *Szóbeli közlés*. Budakalász, 2002. 6.1

[154] A. Iványi: Performance bounds for simple bin packing algorithms. *Annales Univ. Sci. Budapest., Sectio Computatorica* **5** 1984, 77–82. MR 87h:68067, ZB 592.90045. 6.1

[155] A. Iványi: On the d-complexity of words. *Annales Univ. Sci. Budapest. Sectio Computatorica* **8** (1987), 69–90. MR90d:68063, ZB663.68085. 6.1

[156] A. Iványi: Tight worst-case bounds for bin packing algorithms. In: *Theory of Algorithms*. North-Holland, Amsterdam, 1985. 233–240. MR 88g:68051. 6.1

[157] A. Iványi and Z. Tóth: Existence of de Bruijn words. In: *Second Conference on Automata, Languages and Programming Systems* (Salgótarján, 1988), Karl Marx Univ. Econom., Budapest, 1988, 165-172. oldal. MR 91k:05016. 6.1

[158] A. Iványi, Construction of infinite de Bruijn arrays. *Discrete Applied Mathematics* **22** (3) 1988/89, 289–293. MR 80d.68063, ZB 662.9411. 6.1

[159] A. Iványi: Construction of three-dimensional perfect matrices. *Ars Combinatoria* **29C** (1990), 33-40. 6.1

[160] A. Iványi: Reconstruction of interval tournaments. Technical Reports in Informatics of Eötvös Loránd University **2003-A01**. Budapest, 2003, 8 oldal. 6.1

[161] Iványi: Maximal tournaments. *Pure Mathematics and Applications*, 2003. 14 oldal (megjelenőben). 6.1, 6.5

[162] Iványi Antal: *Ütemezéselmélet*. Elektronikus kézirat. ELTE Informatikai Tanszékcsoport, Budapest, 2003. 48 oldal, 15 hivatkozás. 6.5

[163] A. Iványi, I. Kátai: Processing of random sequences with priority. *Acta Cybernetica* **4** (1) 1978/79, 85–101. ISBN 0324-721X. 6.1

[164] A. Iványi, I. Kátai: Modeling of priorityless processing in an interleaved memory with a perfectly informed processor. *Automation and Remote Control* **46** 1985 (4), 520–526. Translation from *Avtom. Telemekh.* (4) 1985, 129–135. ZB 581.68035 6.1

[165] A. Iványi honlapja: szóbonyoltsággal, versenyekkel, soros és párhuzamos algoritmusokkal kapcsolatos irodalomjegyzék és címlista. 6.1, 6.5

[166] Iványi Antalné, Kovács Zoltán: *A PL/1 programozási nyelv*. Tankönyvkiadó, Budapest, 1979. 250 oldal, 6 hivatkozás. 6.5

[167] A. Iványi, J. Pergel: Performance evaluation of an algorithm, processing 0-1 sequences with priority. *Annales Univ. Sci. Budap., Sectio Computatorica* **5** 1984, 37-40. ZB 613.65148 6.1

[168] Iványi A., Szmeljánszkij, R. L.: *Az elméleti programozás elemei* (oroszul). Moszkvai Állami Egyetem, Moszkva, 1985. 193 oldal. 6.1, 6.5

[169] J. Jájá: *An Introduction to Parallel Algorithms: Design and Analysis*. Addison-Wesley, Reading, 1992. X+566 oldal, ISBN 0201548569. 6.1

[170] Jakobi, Gyula: *Az ODRA-1013 számítógép programozása*. Tankönyvkiadó, Budapest, 1987. 122 oldal. 6.5

[171] Járai Antal: *Mérték és integrál*. Nemzeti Tankönyvkiadó, Budapest, 2002. 198 oldal, 50 hivatkozás, ISBN 963 19 3273 7. 6.5

[172] Járai Antal: *Maple feladatok*. Elektronikus feladatgyűjtemény. ELTE Informatikai Tanszékcsoport, Budapest. 12 oldal. 6.1, 6.5

[173] L. A. Jeffress (szerkesztő): *Celebral Mechanisms in Behavior*. John Wiley & Sons, New York, 1951. 266

[174] N. D. Jones: *Computability and Complexity. From a Programming Perspective.* The MIT Press, Cambridge. 1997. 466 oldal, 166 hivatkozás, ISBN 0-262-10064-9. 6.5

[175] D. Jungnickel: *Graphs, Networks and Algorithms*. Springer-Verlag, Berlin, 2002. 588 oldal, 617 hivatkozás, ISBN 3-540-63760-5. 6.2

[176] Jutasi István: *Az Internet felépítése és működése*. Műszaki Könyvkiadó, Budapest, 1997. 104 oldal, ISBN 963 16 1253 8. 6.5

[177] P. Kacsuk, G. Kotsis (szerkesztők): *Distributed and Parallel Systems*. Kluwer Academic Publishers, Dordrecht, 2000. VII+541 oldal, 30 cikk, ISBN 079-237-892-X. 6.1, 6.5

[178] P. Kacsuk, D. Kranzlmüller, Zs. Németh, J. Volkert: *Distributed and Parallel Systems: Cluster and Grid Computing*. Kluwer Academic Publishers, Dordrecht, 2002. *The Kluwer International Series in Engineering and Computer Science*, **706**. 232 oldal, ISBN 1402072090. 6.1, 6.5, 146

[179] Kása Zoltán: *Algoritmusok tervezése*. Studium Könyvkiadó, Kolozsvár, 1994. 120 oldal, 11 hivatkozás, ISBN 973-96342-2-2. 6.1

[180] Kása Zoltán: Gráfelméleti feladatok. Kolozsvár, BBE, 2003. 6.2

[181] Kátai Imre: *Szimulációs módszerek*. Tankönyvkiadó, Budapest, 1981. 163 oldal. 6.5

[182] Katona Y. Gyula: Algoritmuselmélet. Elektronikus előadásjegyzet. BME, Budapest, 2003. 654 oldal. 6.1

[183] Katona Y. Gyula, Recski András, Szabó Csaba: *Gráfelmélet, algoritmuselmélet és algebra*. BME, Budapest, 1997. 235 oldal. 6.2

[184] A. Kaufmann: *Des points et des fleches ... la théorie des graphes*. Magyarul: *Pontok, élek, ívek ... gráfok*. Műszaki Könyvkiadó, Budapest, 1972, 173 oldal, 21 hivatkozás. 6.2

[185] Kelemen József, Nagy Sára: *Bevezetés a mesterséges intelligenciába*. ELTE TTK, Budapest, 1990. 132 oldal, 102 hivatkozás. 6.5

[186] A. Kemnitz, S. Dulff: Score sequences of multitournaments. *Congressus Numerantium* **127** 1997, 85–95. 6.1

[187] B. W. Kernighan, R. Pike: *The UNIX programming Environment*. Prentice-Hall Inc., 1984. Magyarul: *A UNIX operációs rendszer*. Műszaki Könyvkiadó, Budapest, 1987. 362 oldal, ISBN 963 10 7452 8.

[188] B. W. Kernighan, D. M. Ritchie: *The C Programming Language*. Bell Telephone Laboratories, 1988. Magyarul: *A C programozási nyelv* (fordította Molnár Ervin). Műszaki Könyvkiadó, Budapest, 1994. 292 oldal, ISBN 963 16 0552 3.

[189] J. H. Kingston: *Algorithms and Data Structures*. Addison-Wesley, Harlow, 1997. 380 oldal, 47 hivatkozás, ISBN 0-201-40374-9.

[190] Kiss Béla, Krebsz Anna: *Játékelmélet*. SzIF Universitas, Győr, 1999. 72 oldal, 8 hivatkozás. 6.5

[191] Kiss Ottó, Kovács Margit: *Numerikus módszerek*. Műszaki Könyvkiadó, Budapest, 1973. 547 oldal, 7 hivatkozás. ETO 681.3.041. 6.5

[192] Klein Sándor: *The Effects of Modern Mathematics*. Akadémiai Kiadó, Budapest, 1987. 436 oldal, 532 hivatkozás, ISBN 963 05 4023 1. 6.5

[193] Klincsik Mihály, Maróti György: *Maple 8 tételben a matematikai problémamegoldás művészetéről*. Novadat Számítástechn. Bt., Budapest, 1996. 361 oldal, ISBN 9638541725. 6.5

[194] D. E. Knuth: Big omicron and big omega and big theta. *ACM SIGACT News* **8** (2) 1976, 18–23. 6.1

[195] D. E. Knuth, R. L. Graham, O. Patashnik: *Concrete Mathematics: A Foundation for Computer Science*. Második kiadás. Addison-Wesley, Reading, 1995. oldal, 383 hivatkozás, ISBN 0-201-55802-5. Magyarul: *Konkrét matematika* (szerkesztette Kátai Imre). Műszaki Könyvkiadó, Budapest, 1998. 647 oldal, 383 hivatkozás, ISBN 963-16-1422-0. 6.1

[196] D. E. Knuth: *The Stanford Graphbase:A Platform for Combinatorial Computing*. Addison-Wesley, Reading, 1994. 6.5

[197] D. E. Knuth: *MMIXware: a RISC computer for the third millenium*. Springer-Verlag, 1999. VIII+5550 oldal, ISBN 3 540 66938 8. 6.5

[198] D. E. Knuth: *The Art of Computer Programming. Vol. 1. Fundamental Algorithms. Third Edition.* Addison-Wesley, Reading, 1997. 650 oldal, ISBN 0-201-89683-4. Magyarul: *A számítógép-programozás művészete. 1. kötet. Alapvető algoritmusok* (szerkesztő: Simonovits Miklós). Műszaki Könyvkiadó, Budapest, 1987. 654 oldal, 963 10 7156 1. 6.1

[199] D. E. Knuth: *The Art of Computer Programming. Vol. 2. Seminumerical Algorithms*. Addison-Wesley, Reading, 1998. 762 oldal. ISBN 0 201-89684-2. Magyarul: *A számítógép-programozás művészete. 2. kötet. Szeminumerikus algoritmusok* (szerkesztő: Simonovits Miklós). Műszaki Könyvkiadó, Budapest, 690 oldal, ISBN 963 10 7119 7. 6.1

[200] D. E. Knuth: *The Art of Computer Programming. Vol. 3. Searching and Sorting*. Addison-Wesley, Reading, 1998. 780 oldal. ISBN 0 201-89685-0. Magyarul: *A számítógép-programozás művészete. 3. kötet. Keresés és rende-*

zés. (szerkesztő: Simonovits Miklós). Műszaki Könyvkiadó, Budapest, 1988 és 1994.761 oldal, ISBN 963 16 077 7. 6.1

[201] D. E. Knuth: *Selected Papers on Analysis of Algorithms*. Center for the Study of Language and Information, Leland Stanford Junior University, 2000. 621 oldal, 408 hivatkozás, ISBN 1-57586-212-3. 6.1

[202] D. E. Knuth: *The Art of Computer Programming. Vol. 4. Combinatorial Algorithms*. Elektronikus kézirat. Stanford University, 2003. 208 oldal. 6.1, 6.5

[203] Komlósi Sándor (szerkesztő): *Az optimalizáláselmélet alapjai*. Dialóg Campus Kiadó, Budapest-Pécs, 2001. 408 oldal, ISBN 963 9123 09 9. 6.5

[204] E. Korach, S. Moran, S. Zaks: The optimality of distributive constructions of minimum weight and degree restricted spanning trees in a complete network of processors. In: *Proceedings of Symposium on Principles of Distributed Computing*. 1985.

[205] Korolev, L. N.: *Számítógépek felépítése és szoftverje* (oroszul). Nauka, Moszkva, 1978. 351 oldal, 29 hivatkozás.

[206] Kotov, V. E. *Petri-hálók*. Nauka, Moszkva, 1984. 158 oldal, 89 hivatkozás. 6.5

[207] Kovács Attila: Komputeralgebra a tudományokban és a gyakorlatban. *Alkalmazott Matematikai Lapok* **18** 1998, 181–202. 66 hivatkozás.

[208] A. Kovács: Computer Algebra: Impact and Perspectives. *Nieuw Archief voor Wiskunde* **17** (1) 1999, 29-55.

[209] E. Kovács, Z. Winkler: *5th International Conference on Applied Informatics*. Molnár és társa, Eger, 2002. 250 oldal, 30 cikk. 87

[210] Kovács Gábor Zsolt, Pataki Norbert: Rangsorolási algoritmusok. TDK dolgozat. ELTE, Informatikai Tanszékcsoport, Budapest, 2002. 6.1

[211] Kovács Győző: *A számítógépek technikája*. Tankönyvkiadó, Budapest, 1974. 280 oldal, 15 hivatkozás, ISBN 963 17 0531 5. 6.5

[212] Kovács Margit: *A nemlineáris programozás elmélete*. Typotex, Budapest, 1997. 200 oldal, ISBN 9637546855. 6.5

[213] Kovács Margit: *Operációkutatás II*. Elektronikus jegyzet. ELTE TTK Operációkutatási Tanszék, Budapest, 2000. 6.5

[214] M. Kovács, F. P. Vasziljev, R. Fullér: Stability of a fuzzy solution of systems of linear algebraic equations with fuzzy coefficients. *Moscow University Computational Mathematics & Cybernetics* **15** (1) 1989, 4–9; translated from: *Proceedings of the Moscow State University*. **15** (1) 1989, 5–9 (in Russian). MR 91c:94039, ZB 681.65028.

[215] Kozics Sándor: *Az ALGOL60, a FORTRAN, a COBOL és a PL/I programozási nyelvek*. ELTE TTK, Budapest, 1992. 246 oldal, 52 hivatkozás. 6.5

[216] Kozma László, Varga László: *Párhuzamos rendszerek elemzése*. ELTE Informatikai Tanszékcsoport, Budapest, 2002. 151 oldal, 72 hivatkozás. 6.5

[217] Kozsik Tamás: The application of an associative function on the prefixes of a series. Technical Reports in Informatics of Eötvös Loránd University **2003-A02**. Budapest, 2003, 22 oldal. 6.2

[218] Kőhegyi János (szerkesztő): *Ismerd meg a BASIC nyelvjárásait!* (*HT–1080Z, ABC80, ZX–81*) Műszaki Könyvkiadó, Budapest, 1984. 164 oldal, ISBN 963 10 6183 3. 6.5

[219] Kőhegyi János (szerkesztő): *Ismerd meg a BASIC nyelvjárásait!* (*ZX Spectrum, TI99/4A, PROPER–16*) Műszaki Könyvkiadó, Budapest, 1985. 186 oldal, ISBN 963 10 6183 3 6.5

[220] Kőhegyi János (szerkesztő): *Ismerd meg a BASIC nyelvjárásait!* (*Commodore 64, Commodore VIC 20, SHARP PC-1500*) Műszaki Könyvkiadó, Budapest, 1986. 185 oldal, ISBN 963 10 7051 4. 6.5

[221] Kőhegyi János (szerkesztő): *Ismerd meg a BASIC nyelvjárásait!* (*Commodore 16, Commodore PLUS4, Commodore 128, VIDEOTON TV-

Computer) Műszaki Könyvkiadó, Budapest, 1986. 235 oldal, ISBN 963 10 8155 9. 6.5

[222] Kurtán Lajos: A közgazdaságtan alapjai. ELTE Eötvös Kiadó, Budapest, 1996. 274 oldal. 6.5

[223] Kurtán Lajos: Piacgazdaságtan. ELTE Eötvös Kiadó, Budapest, 2001. 6.5

[224] T. H. Lai, S. Sahni: Anomalies in parallel branch and bound algorithms. *Communications of* ACM 1984, 594–602. 6.1

[225] Láng Csabáné: *Bevezetés a matematikába. I.*. ELTE TTK, Budapest, 1994. 6.1, 6.5

[226] Láng Csabáné, Gonda János: *Bevezetés a matematikába. II*. ELTE TTK, Budapest, 1995. 240 oldal, 18 hivatkozás. 6.1, 6.5

[227] W. B. Langdon, R. Poli: *Foundations of Genetic Programming*. Springer-Verlag, New York, 2002. 274 oldal, ISBN 3540424512. 6.5

[228] I. A. Lavrov, L. L: Maximova: *Halmazelméleti, matemaikai logikai és algoritmuselméleti feladatok*. Oroszul Nauka, Moszkva, 1984. magyarul: Műszaki Könyvkiadó, Budapest, 1987 (lektorálta: Urbán János). 223 oldal, 38 hivatkozás, ISBN 963 10 7240 X.

[229] E. L. Lawler: *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart, and Winston, 1976. Magyarul: *Kombinatorikus optimalizálás* (lektorálta Katona Gyula). Műszaki Könyvkiadó, Budapest, 1982. 358 oldal. ISBN 963 10 4181 6. 6.1

[230] LEDA: LEDA (Library of Efficient Data Types and Algorithms. Algorithmic Solutions Software GmbH, 2003 (karbantartva, fizetős). 6.1

[231] D. H. Lehmer: A machine for combining sets of linear congruences. *Mathematical Annalen* **109** (1934), 661-667. 6.1

[232] D. H. Lehmer, A photo-electric number sieve. *American Mathematical Monthly* **40** (1933), 401-406. 6.1

[233] T. F. Leighton: *Introduction to Parallel Algorithms and Architectures: Arrays-Trees-Hypercubes*. Morgan Kaufmann, San Francisco, 1992. 831+XX oldal, ISBN 1-55860-117-1. 6.1, 6.3, 6.4

[234] T. F. Leighton: *Introduction to Parallel Algorithms and Architectures: Algorithms and VSLI*. Morgan Kaufmann Publishers, San Francisco, 2001. 6.1, 6.4

[235] G. LeLann: *Distributed Systems: Towards a Formal Approach*. In: [110], 155–160. 6.5

[236] C. Leopold: *Parallel and Distributed Computing: A Survey of Models, Paradigms and Approaches*. John Wiley & Sons. 2000. 272 oldal, 457 hivatkozás, ISBN 0471358312. 6.1

[237] M. Li, P. Vitanyi: *An Introduction to Kolmogorov Complexity and its Applications*. Springer-Verlag, New York, 1997. XX+637 oldal, 519 hivatkozás, ISBN 0-387-94868-6. 6.5

[238] A. A. Levitin: *Introduction to the Design and Analysis of Algorithms*. Addison-Wesley Educational Publishers, Inc., 2002. 528 oldal, ISBN 0201743957.

[239] L. Lovász: *Combinatorial Problems and Exercises*. Akadémiai Kiadó, Budapest, 1979. 551 oldal, 20 hivatkozás. ISBN 963 05 1469 9. magyarul: *Kombinatorikai problémák és feladatok*. Typotex, Budapest, 1999. 666 oldal, 39 hivatkozás, ISBN 963 9132 37 3. 6.5

[240] Lovász László: *Algoritmusok bonyolultsága*. ELTE TTK, Budapest, 1990. 134 oldal, 12 hivatkozás. 6.5

[241] Lovász László, Gács Péter: *Algoritmusok*. Műszaki, Budapest, 1978. 179 oldal, ISBN 963 10 2067 3. Tankönyvkiadó, 1987. 179 oldal, ISBN 963 180 334 1. 6.1

[242] Lőcs Gyula: *Az ALGOL 60 programozási nyelv*. 5. kiadás. Műszaki Könyvkiadó, Budapest, 1973. 254 oldal, 24 hivatkozás. 6.5

[243] Lőcs Gyula, Vigassy József: *A FORTRAN programozási nyelv*. 3. kiadás. Műszaki Könyvkiadó, Budapest, 1973. 335 oldal, 14 hivatkozás. 6.5

[244] É. Lucas: *Récréations mathématiques*. 4 kötet. Gauthier-Villars, Paris, 1891-1894. 6.1

[245] N. A. Lynch: *Distributed Algorithms*. Ötödik kiadás. Morgan Kaufman Publishers, San Francisco, 2001, 876 oldal, 290 hivatkozás, ISBN 1-55860-384-4. (Magyarul: *Osztott algoritmusok* (szerkesztette: Iványi Antal). Kiskapu Kiadó, Budapest, 2002. 781 oldal, 290 hivatkozás, ISBN 963 9301 03 5. 6.1, 6.5

[246] V. Malyshkin (szerkesztő): *Parallel Computing Technologies*. *LNCS* **1662**. Springer-Verlag, Berlin, 1999. 510 oldal, 59 cikk. ISBN 3-540-666363-0. 6.5

[247] Zs. Marczell, Cs. Szepesváry, Zs. Kalmár, A. Lőrincz: Parallel and robust skeletonization built from self-organizing elements. *Neural Networks* **12** 1999, 163–173. 30 hivatkozás. 6.5

[248] Marton László, Fehérvári Arnold: *Algoritmusok és adatstruktúrák*. Novadat, Győr, 2002. 344 oldal, 8 hivatkozás, ISBN 963 9056 33 2. 6.1

[249] E. W. Mayr, H. J. Prömel, A. Steiger: *Lectures on Proof Verification and Approximation Algorithms. LNCS* **1367**, Springer-Verlag, Berlin, 1998. 344 oldal, 133 hivatkozás. ISBN 3-540-64201. 6.1

[250] K. Mehlhorn: *Data Structures and Algorithms. Volume 1. Sorting and Searching*. Springer-Verlag, Berlin, 1984. ISBN 3-540-13302-X. 6.1

[251] K. Mehlhorn: *Data Structures and Algorithms. Volume 2. Graph Algorithms and NP-Completeness*. Springer-Verlag, Berlin, 1984. ISBN 3 540 13641-X. 6.1

[252] K. Mehlhorn: *Data Structures and Algorithms. Volume 3. Multidimensional Searching and Computational Geometry*. Springer-Verlag, Berlin, 1984. ISBN 3-540-13642-8. 6.1

[253] LEDA: LEDA (Library of Efficient Data Types and Algorithms. Algorithmic Solutions Software GmbH, 2003 (karbantartva, fizetős). 6.1

[254] R. Miller, L. Boxer: *Algorithms Sequential and Parallel: A Unified Approach*. Prentice Hall, 2000, 336 oldal, 61 hivatkozás. ISBN 0-13-086373-4. 6.1

[255] J. Misra: *A Discipline of Multiprogramming: Programming Theory for Distributed Applications*. Springer-Verlag, Nerw York, 2001. 440 oldal, ISBN 0387952063. 6.5

[256] M. Mitchell: *An Introduction to Genetic Algorithms*. The MIT Press, 1998. 224 oldal, ISBN 0262631857. 6.5

[257] Molnárka Győző, Gergó Lajos, Wettl Ferenc, Horváth Arnold, Kallós Gábor: *A Maple V és alkalmazásai*. Springer Hungarica Kiadó, Budapest, 1996. 330 oldal, 18 hivatkozás, ISBN 963 8455 89 6. 6.5

[258] J. W. Moon: *Topics on Tournaments*. Holt, Rinehart and Winston. New York, 1968. 103 oldal, 146 hivatkozás. 6.1

[259] O. Morgenstern, J. Neumann: *Theory of Games and Economic Behavior*. Princeton University Press, Princeton, 1944, 1947 és 1953. XVIII+625, 641 és 641 oldal. 6.5

[260] R. Motwani, P. Raghavan: *Randomized Algorithms*. Cambridge University Press, Cambridge, 2000. 476 oldal, 424 hivatkozás. ISBN 0 521 47465 5. 6.1

[261] T. V. Narayana, D. H. Bent: Computation of the number of score sequences. *Canadian Mathematical Bulletin* **7** 1964, 133–136. ISBN 0008-4395. 6.1

[262] Nagy Tibor: *Programanimációk*. Elektronikus kézirat. Budapest, 2003. ELTE Informatikai Tanszékcsoport. 29 animált algoritmus. 6.5

[263] D. Nassimi, S. Sahni: Parallel permutation and sorting algorithms and a generalized interconnection network. *Journal of the ACM* **29** (3) (1982), 642–667. 6.3

[264] Netlib: *Collection of Mathematical Software, Papers, and Databases*. The University of Tennesse és Oak Ridge National Laboratory, 2003. 6.1

[265] J. Neumann: *The Computer and the Brain*. Yale University Press, New Haven, 1958. Magyarul: *A számítógép és az agy*. Gondolat, Budapest, 1972. 133 oldal. 6.1

[266] J. Neumann: *General and Logical Theory of Automata*. In: [173], 1–31 és in [269], 288–328. 6.1

[267] J. Neumann: *Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components*. In: [331], 43–98 (lásd még: in [**?**]). 6.1

[268] J. Neumann: *The Computer and the Brain*. Yale University Press. 1958. 82 oldal. 6.1

[269] J. Neumann: *Collected papers. Volume 5. Design of Computers, Theory of Automata and Numerical Analysis*. Pergamon Press, Oxford, 1963. 784 oldal. 266

[270] J. Neumann: *Theory of Self-reproducing Automata* (szerkesztő: W. Burks). University of Illinois Press, Urbana, 1966. XIX+388 oldal, 81 hivatkozás. Oroszul: *Az önreprodukáló automaták elmélete*. Moszkva, Mir, 1971. 81 hivatkozás, 382 oldal. 6.1

[271] A. Nijenhuis, H. Wilf: *Combinatorial Algorithms*. Academic Press, New York, 1978. 302 oldal, 39 hivatkozás, ISBN 0-12-519260-6. 6.5

[272] P. Norton, M. Stockman: *Network Security Fundamentals*. Magyarul: *A hálózati biztonság alapjairól*. Kiskapu, Budapest, 2000. 501 oldal, 963 93012 1 3. 6.5

[273] Nyékyné Gaizler Judit (szerkesztő): *Az Ada programozási nyelv*. ELTE Eötvös Kiadó, Budapest, 1998. 576 oldal, 25 hivatkozás, ISBN 963 463 238 6. 6.5

[274] Nyékyné Gaizler Judit (szerkesztő): *J2EE útikalauz Java programozóknak. + CD-melléklet*. ELTE TTK Hallgatói Alapítvány, Budapest, 2002. 719 oldal, 146 hivatkozás, ISBN 963 463 578 4. 6.5

[275] Nyékyné Gaizler Judit (szerkesztő): *Programozási nyelvek*. Kiskapu Kiadó, Budapest, 2003. 800 oldal, 192 hivatkozás, ISBN 963 9301 477. 6.5

[276] Obádovics J. Gyula: *Numerikus módszerek és programozásuk*. Tankönyvkiadó, Budapest, 1975. 304 oldal, 48 hivatkozás, ISBN 963 17 6877 2. 6.5

[277] Obádovics J. Gyula: *Matematika*. 17. kiadás. Scolar, Budapest, 2002. 816 oldal, ISBN 9639193704.

[278] Officina Nova: *Képes világatlasz. A kék bolygó. A Föld és a természet*. Officina, Budapest, 2001. 376 oldal, ISBN 963 477 037 1. 6.1

[279] K. Okuguchi, F. Szidarovszky: *The Theory of Oligopoly with Multi-Product Firms*. Spinger-Verlag, New York, 1990 és 1999. 268 oldal, 138 hivatkozás, ISBN 3-540-65779-7. 6.5

[280] O. Ore: *Graphs and their Uses*. Random House, New York, 1944. Magyarul: *A gráfok és alkalmazásaik*. Gondolat Kiadó, Budapest, 1972. 158 oldal. 6.2

[281] J. Pachl, E. Korach, D. Rotem: Lower bounds for distributed maximum finding algorithms. *Journal of* ACM **31** 1984, 905-918. 6.5

[282] C. H. Papadimitriou: *Computational Complexity*. Addison-Wesley, 1995. Magyarul: *Számítási bonyolultság* (szerkesztette: Ésik Zoltán). Novadat, Győr, 1999. 589 oldal. ISBN 963 9056 20 0. 6.1, 6.5

[283] C. H. Papadimitriou, K. Steiglitz: *Combinatorial Optimization: Algorithms and Complexity*. Dover Publ., Mineola, 1998. 496 oldal, 380 hivatkozás. ISBN 0-486-40258-4.

[284] P. M. Pardalos; S. Rajasekaran (szerkesztők): *Advances in Randomized Parallel Computing*. Kluwer Academic Press Publ., Dordrecht, 1999. 352 oldal, ISBN 0-7923-57140. 6.1

[285] P. M. Pardalos, M. G. C. Resende: *Handbook of Applied Optimization*. Oxford University Press, Inc. New York, 2002. XVIII+1095 oldal, ISBN 0195-125940. 6.5, 49

[286] R. Parent: *Computer Animation. Algorithms and Techniques*. Morgan Kaufmann Publishers, San Francisco, 2002. XXII+527 oldal, 347 hivatkozás, ISBN 1-55860-579-7. 6.1, 6.5

[287] Pásztorné Varga Katalin, Várterész Magdolna: *Matematikai logika alkalmazásszemléletű tárgyalása*. Panem, Budapest, 2003. 350 oldal, 45 hivatkozás, ISBN 963 545 694 7. 6.5

[288] G. Pécsy, L. Szűcs: Parallel verification and enumeration of tournaments. *Studia Univ. Babes-Bolyai Informatica* **XLV** (2) 2000, 11–26. 6.1

[289] G. L. Peterson: An $O(n \log n)$ unidirectional algorithm for the circular extrema problem. *ACM Transactions on Programming Languages and Systems* **4** 1982, 758–762. 6.5

[290] G. L. Peterson: Efficient algorithms for elections in meshes and complete networks. *Technical Reports* **TR140**. Department of Computer Science, University of Rochester, Rochester, 1985. 6.5

[291] A. Pethő: *Algebraische Algorithmen*, Vieweg Verlag, 1999. 234 oldal, ISBN: 3-528-06598-2. 6.5

[292] Pirkó József: *Turbo Pascal 6.0 for Windows*. LSI Kiadó, Budapest, 1992. 682 oldal, ISBN 963 577 021 9. 6.5

[293] Pirkó József: *Turbo Pascal 7.0*. LSI Kiadó, Budapest, 1993. 222 oldal, ISBN 963 577 059 6. 6.5

[294] Pongor György: *Pascal. Programozás és algoritmusok*. Novotrade Rt., Budapest, 1988. 440 oldal, 13 hivatkozás, ISBN 963 02 58978. 6.5

[295] R. E. Prather: *Elements of Discrete Mathematics*. Houghton Mifflin Company, Boston, 1986. 538 oldal, ISBN 0-395-35165-0. 6.5

[296] Prékopa András: Valószínűségelmélet műszaki alkalmazásokkal. Műszaki Könyvkiadó, Budapest, 1962. 440 oldal, 30 hivatkozás, ISBN 963 10 0575 5. 6.1

[297] F. P. Preparata: New parallel sorting schemes. *IEEE Transactions on Computers* **C-27** (7) 1978, 669–673. 6.2

[298] P. W. Purdom, C. A. Brown: *The Analysis of Algorithms*. Holt, Rinehart and Winston, New York, 1985. 540 oldal, 141 hivatkozás, ISBN 0-03-072044-3. 6.1

[299] I. C. Pyle: *The Ada Programming Language: a Guide for programmers*. 2. kiadás. Prentice-Hall, Englewood Cliffs, 1985. X+341 oldal, ISBN 0-13-003-906-3. Magyarul: *Az Ada programozási nyelv* (szerkesztette: Zajki László). Műszaki Könyvkiadó, Budapest, 1987. 310 oldal, ISBN 963 107281 9. 6.5

[300] M. J. Quinn: *Designing and Analysing of Parallel Algorithms*. Oxford University Press, New York, 1993. 288 oldal, 498 hivatkozás, ISBN 0-07-051071-7.

[301] S. Rajasekaran, S. Sen: Random sampling techniques and parallel algorithms design. In: [309], 411–451.

[302] S. Rajasekaran, T. Tsantilas: Optimal routing algorithms for mesh-connected processor arrays. *Algorithmica* **8** (1992), 21:28. 6.3

[303] S. Rajasekaran: *Computer programs*. University of Florida, Florida, 2003. 6.1

[304] A. Ralston: *Introduction to Programmig and Computer Science*. McGraw-Hill, Inc.. Magyarul: *Programozás és számítógéptudomány*. Műszaki Könyvkiadó, Budapest, 1974. 575 oldal, 95 hivatkozás, ISBN 96310 0616 6.

[305] A. Ralston: *A First Course in Numerical Analysis*. McGraw-Hill. Magyarul: *Bevezetés a numerikus analízisbe* (lektorálta Kiss Ottó). Műszaki Könyvkiadó, Budapest, 1969. 572 oldal, 282 hivatkozás. 6.5

[306] A. Ralston, E. D. Reilly, D. Hemmendinger (szerkesztők): *Encyclopedia of Computer Science*. Negyedik kiadás. Nature Publishing Group, London, 2000. 2034 oldal. ISBN 1-561-59248-X. 6.1

[307] S. Ranka, S. Sahni: *Hypercube Algorithms with Applications to Image Processing and Pattern Recognition*. Springer-Verlag, Bilkent University Lecture Series, 1990. 6.4

[308] Recski András: *Matroid Theory and its Applications in Electric Networks and Theory of Statistics*. Akadémiai Kiadó, Budapest, 1989. XIII+531 oldal, 422 hivatkozás, 963 05 5253 1. 6.2

[309] J. H. Reif (szerkesztő): *Synthesis of Parallel Algorithms*. Morgan Kaufmann, San Mateo, 1992. VI+1011 oldal, ISBN 1 558 601 135 X. 301

[310] J. H. Reif, L. Valiant: *A logarithmic time sort for linear size networks*. *Journal of* ACM **34** (1) 1987, 60–76.

[311] R. Reischuk: Probabilistic parallel algorithms for sorting and selection. *SIAM Journal on Computing* **14** (2) 1989, 594–607. 6.2

[312] A. Rényi: *Probability Theory*. Akadémiai Kiadó/North Holland Publ. House, Budapest/Amsterdam, 1970. 666 oldal, 507 hivatkozás. Magyarul: *Valószínűségelmélet*. Tankönyvkiadó, Budapest, 1973. 510 oldal, 492 hivatkozás. 6.1

[313] F. Rolland: *The Essence of Databases*. Prentice Hall, 1998. Magyarul: Adatbázisrendszerek (lektorálta Kiss Attila). Panem, Budapest, 2000. 236 oldal, 41 hivatkozás, ISBN 963 545 348 5. 6.5

[314] Rónyai Lajos, Ivanyos Gábor, Szabó Réka: *Algoritmusok*. Typotex, Budapest, 1999. 349 oldal, ISBN 963 9132 16 0. 6.1

[315] S. H. Roosta: *Parallel Processing and Parallel Algorithms*. Springer-Verlag, New York, 2000. 566 oldal, 373 hivatkozás. ISBN 0-387-98716-9. 6.1

[316] T. Roska, A. Rodríguez-Vázquez (szerkesztők): *Towards the Visual Microprocessor: VLSI Design and the Use of Cellular Neural Network Universal Machine*. John Wiley & Sons, Chichester, 2001. 400 oldal, ISBN 0471956066. 6.5

[317] S. J. Russell, P. Norwig: *Artificial Intelligence. A Modern Approach*. Prentice-Hall International, Englewood Cliffs, 1995. 932 oldal, 1174 hivatkozás, ISBN 0-13-360124-2. Magyarul: *Mesterséges intelligencia modern megközelítésben*. Panem, Budapest, 2000. 1093 oldal, 1174 + 93 hivatkozás, ISBN 963 545 241 1. 6.5

[318] Ruzsa Imre: *Bevezetés a modern logikába*. Osiris, Budapest, 2001. 391 oldal, 98 hivatkozás, ISBN 963 379 978 3. 6.5

[319] W. Rytter: *Jewels of Stringology*. World Scientific Publ. Comp., 2002. X+310 oldal, 152 hivatkozás, ISBN 981 024782 6.

[320] L. E. Sadovskii, A. L. Sadovskii: *Mathematics and Sports*. American Mathematical Society, Providence, 1993. 191 oldal. 6.1

[321] Salamon Gábor: *Algoritmusok animációja*. Budapest, BME, 2003. 40 program. 6.5

[322] J.-R. Sack, J. Urrutia (szerkesztők): *Handbook of Computational Geometry*. North-Holland, Amsterdam, 2000. X+1027+48 oldal. ISBN 0-444-82537-1. 6.3

[323] S. Salleh, A. Y. Zomaya: *Scheduling in Parallel Computing Systems: Fuzzy and Annealing Techniques*. Kluwer, 1999. ISBN 0-7923-8533-0.

[324] A. Salomaa: *Theory of Automata*. Pergamon Press, Oxford, 1969.

[325] A. Salomaa, G. Rozenberg, W. Brauer (szerkesztők): *Public-Key Cryptography*. Springer-Verlag, New York, 2001. 271 oldal, ISBN 3540613560. 6.5

[326] Sántáné Tóth Edit: *Tudásalapú technológia, szakértő rendszerek*. Második kiadás. Miskolci Egyetem Dunaújvárosi Főiskolai Kar Kiadó Hivatala, Dunaújváros, 2001. 301 oldal, 162 hivatkozás. 6.5

[327] F. Schipp, W. R. Wade, P. Simon: *Walsh Series. An Introduction to Dyadic Harmonic Analysis* (with the collaboration of J. Pál). Adam Hilger Ltd., Bristol, 1990. X+560 oldal, 383 hivatkozás, ISBN 963 05 58807. MR 92g:42001 6.5

[328] U. Schöning: *Algorithmik*. Spektrum Akademischer Verlag, Heidelberg, 2001. 384 oldal, 154 hivatkozás, ISBN 3-8274-1092-4. 6.1

[329] R. Sedgewick, P. Flajolet: *Analysis of Algorithms*. Addison-Wesley Publ. Company, Reading, 1996. 492 oldal, 240 hivatkozás, ISBN 0-201-40009-X. 6.1

[330] G. A. Selim: *The Design and Analysis of Parallel Algorithms*. Prentice-Hall, 1989. 400 oldal, ISBN 013 200056 3. 6.1

[331] C. E. Shannon, J. McCarthy: *Automata Studies*. Princeton University Press, 1956. 267

[332] Siklósi Bence: Soros és párhuzamos algoritmusok összehasonlítása sportversenyekkel kapcsolatos problémákban. Diplomamunka. ELTE Informatikai Tanszékcsoport, Budapest, 2001. 6.1

[333] A. Silberschatz, P. Galvin, G. Gagne: *Applied Operating Systems Concepts*. John Wiley & Sons, New York, 2000. 840 oldal, 327 hivatkozás, ISBN 0-471-36508-4. 6.5

[334] D. Sima: *Computational models*. Kézirat, Budapest, 1995. 87 oldal. 6.1

[335] D. Sima, T. Fountain, P. Kacsuk: *Advanced Computer Architectures: a Design Space Approach*. Addison-Wesley, Harlow, 1997 és 1998. 766 oldal, 152 hivatkozás, ISBN 0-201-42291-3. Magyarul: *Korszerű számítógéparchitektúrák tervezésitér-megközelítésben*. Szak Kiadó, Bicske, 1998. 809 oldal, 540 hivatkozás, ISBN 963-9131-09-1. 6.1

[336] Simon Péter: *Ismerkedés a numerikus analízissel* (lektorálta Schipp Ferenc és Száva Géza). ELTE TTK Továbbképzési Csoport, Budapest, 1990. 232 oldal, 11 hivatkozás, ISBN 963 462 5614. 6.5

[337] Sipos Annamária: *A Visual* C++ és az MFC. Budapest, 2000. 433 oldal, 27 hivatkozás, ISBN 963 640 979 X. 6.5

[338] M. Sipser: *Introduction to the Theory of Computation*. PWS Publishing Company, 1997. IX+396 oldal, 71 hivatkozás, ISBN 053494728X. 6.5

[339] S. S. Skiena: *The Algorithm Design Manual*. Springer-Verlag, New York, 1998. 486 oldal, 845 hivatkozás, ISBN 0-387-94860-0. 6.1

[340] W. Schreiner: *Osztott algoritmusok*. Elektronikus feladatgyűjtemény, Linz, 2003. 6.1

[341] J. Sommerville: *Software Engineering*. Addison-Wesley, Boston, 1998, 742 oldal, 343 hivatkozás, ISBN 0201-42765-6. Magyarul: *Szoftverrendszerek fejlesztése* (szerkesztette: Juhász István, lektorálta Kormos János). Panem, Budapest, 2001. 752 oldal, 343 hivatkozás, ISBN 963 54531 1 6. 6.5

[342] J. Spencer: *Ten Lectures on the Probabilistic Method*. Regional Conference Series on Applied Mathematics (No. 52). SIAM, Philadelphia, 1987. 78 oldal. 6.1

[343] R. P. Stanley: *Enumerative Combinatorics. Volume 1*. Cambridge University Press, Cambridge, 1999. 326 oldal, 39 hivatkozás, ISBN 0 521 66351 2. 6.5

[344] R. P. Stanley: *Enumerative Combinatorics. Volume 2*. Cambridge University Press, Cambridge, 1999. 585 oldal, 22 hivatkozás, ISBN 0 521 78987 7. 6.5

[345] Stoyan Gisbert: *Matlab. 4. és 5. verzió*. Typotex, Budapest, 1999. 308 oldal, 26 hivatkozás, ISBN 963-9132-33-0. 6.5

[346] Stoyan Gisbert, Takó Galina: *Numerikus módszerek. 1. kötet*. Typotex, Budapest, 1993. 440 oldal, 110 hivatkozás, ISBN 963 9326 20 8. 6.5

[347] Stoyan Gisbert, Takó Galina: *Numerikus módszerek. 2. kötet*. Typotex, Budapest, 1995. 320 oldal, 142 hivatkozás, ISBN 963 7546 53 7. 6.5

[348] Stoyan Gisbert, Takó Galina: *Numerikus módszerek. 3. kötet*. Typotex, Budapest, 1997. 486 oldal, 244 hivatkozás, ISBN 963 7546 77 4 . 6.5

[349] B. Stroustrup: *The* C++ *Programming Language*. *The* C++ *Programming Language* (Special Edition). Addison-Wesley. Reading, 2000. 1029 oldal, 43 hivatkozás, ISBN 0-201-70073-5. Magyarul: *A* C++ *programozási nyelv* (lektor: Porkoláb Zoltán. Kiskapu Kiadó, Budapest, 2001. XXII+1305+21 oldal, 43 hivatkozás, ISBN 963 9301 17 5 ö. 6.1, 6.5

[350] Szántai Tamás: *Új utak a magyar operációkutatásban*. Dialóg Campus Kiadó, Budapest, 1995. 396 oldal, ISBN 963 9123 919. 6.5

[351] Szántai Tamás: *Az operációkutatás matematikai módszerei*. BME, Budapest, 2001. 41 oldal. 6.5

[352] Szeberényi Imre: *Parallel programozás áttekintése*. Elektronikus kézirat, BME, Budapest, 2003. 6.5

[353] Szidarovszky Ferenc: *Bevezetés a numerikus módszerekbe*. Közgazdasági és Jogi Kiadó, Budapest, 1974. 389 oldal, 36 hivatkozás, ISBN 963 220 029 2. 6.5

[354] F. Szidarovszky, A. T. Bahill: *Linear Systems Theory*. CRC Press, Boca Raton, 1992 és 1997. 508 oldal, 442 hivatkozás, ISBN 0-8493-1687-1. 6.5

[355] F. Szidarovszky, S. Yakowitz: *Introduction to Numerical Computations*. Macmillan, New York, 1986 és 1989. XVI+462 oldal, ISBN 00243-08218. 6.5

[356] Szili László, Tóth János: *Matematika és Mathematica*. ELTE Eötvös Kiadó, Budapest, 1996. 396 oldal, 90 hivatkozás, ISBN 963 463 004 9. 6.5

[357] Szirmay-Kalos László: *Számítógépes grafika*. Computerbooks, Budapest, 1999. 334 oldal, 127 hivatkozás, ISBN 963 618 208 6. 6.5

[358] Szlávi Péter, Zsakó László: *Módszeres programozás*. Műszaki Könyvkiadó, Budapest, 1986. 116 oldal, 12 hivatkozás, ISBN 963 10 6820 X 6.5

[359] A. S. Tanenbaum: *Computer Networks*. Pearson Education, 2002. 912 oldal, ISBN 0130661023. Magyarul: *Számítógép-hálózatok*. Panem, Budapest, 2001, 888 oldAL, ISBN 9635452136. 6.5

[360] A. S. Tanenbaum: *Structured Computer Organization*. Prentice Hall International, Upper Sadle River, 1999. XVII+670 oldal, 147 hivatkozás, ISBN 0-13-020435-8. Magyarul: *Számítógépek felépítése* (szerkesztette Csirik János). Panem, 1999, 669 oldal, 147 hivatkozás, ISBN 963 545 2829. 6.5

[361] A. S. Tanenbaum: *Modern Operating Systems*. Második kiadás. Pearson Publisher, 2001. 976 oldal, ISBN 0130313580. 6.5

[362] A. S. Tanenbaum, A. S. Woodhull: *Operating Systems. Design and Implementation*. Prentice Hall, Upper Saddle River, 1997. 939 oldal, 89 hivatkozás, ISBN 0-13-638677-6. Magyarul: *Operációs rendszerek* (szerkesztette Csirik János). Panem, Budapest, 1999. 980 oldal, 89 hivatkozás, ISBN 963 545 189 X. 6.5

[363] A. S. Tanenbaum, M. van Steen: *Distributed Systems. Principles and Paradigmas.* Prentice Hall, Upple Saddle River, 2002. XXIII+813 oldal, 504 hivatkozás, ISBN 0-13-088893-1. 6.5

[364] G. Tel: *Introduction to Distributed Algorithms*. Cambridge University Press, Cambridge, 2001, 596 oldal, 228 hivatkozás, ISBN 0 521 79483 8. 6.1, 6.2, 6.5

[365] B. A. Trahtenbrot: *Algoritmusok és absztrakt automaták*. Műszaki Könyvkiadó, Budapest, 1978. 207 oldal, 16 hivatkozás, ISBN 963 10 1755. 6.1, 6.5

[366] Turcsányiné Szabó Márta: *A LOGO programozási nyelv*. Műszaki Könyvkiadó, Budapest, 1986. 317 oldal, 5 hivatkozás, ISBN 963-106-841-2.

[367] J. D. Ullman, J. Widom: *A First Course in Database Systems*. Prentice Hall Inc., Upper Saddle River, 1997. Magyarul: *Adatbázisrendszerek. Alapvetés* (szerkesztette Benczúr András). Panem, Budapest, 1998. 507 oldal, 38 hivatkozás, ISBN 963 545 190 3. 6.5

[368] G. Valiente: *Algorithms on Trees and Graphs*. Springer-Verlag, New York, 2002. 450 oldal, ISBN 354 043550 6. 6.1

[369] J. Van Leeuwen (szerkesztő): *Handbook of Theoretical Computer Science A: Algorithms and Complexity*. Második kiadás. 2 kötet (A és B). Elsevier, 1990. The MIT Press, Amsterdam. ISBN 0 26 222 0407. 6.1

[370] Varga László: *Rendszerprogramok elmélete és gyakorlata*. Akadémiai Kiadó, Budapest, 1980. 566 oldal, 198 hivatkozás, ISBN 963 05 2296 9. 6.5

[371] Varga László: *Programok analízise és szintézise*. Akadémiai Kiadó, Budapest, 1981. 304 oldal, 141 hivatkozás, ISBN 963 052682 4. 6.5

[372] Varga László, Sike Sándor: *Szoftvertechnológia és UML*. ELTE Eötvös Kiadó, Budapest, 2002. 302 oldal, 10 hivatkozás, ISBN 963 463 477 X. 6.5

[373] U. Vazirani: *Approximation Algorithms*. Springer-Verlag, Berlin, 2001. 378 oldal, 263 hivatkozás, ISBN 3-540-65367-8. 6.5

[374] Vida János honlapja: innen letölthető G. E. Farin könyve [76]. 6.5

[375] N. Vilenkin: *Combinatorial Mathematics for Recreation*. Mir Publisher, Moscow, 1972. 207 oldal. 6.5

[376] Vizvári Béla: *Egészértékű programozás*. Tankönyvkiadó, Budapest, 1990. 6.5

[377] Vizvári Béla: *Bevezetés a termelésirányítás matematikai módszereibe*. ELTE TTK, Budapest, 1994. 262 oldal, 143 hivatkozás. 6.5

[378] J. Vogel, K. Wagner: *Komplexität, Graphen und Automaten*. Friedrich Schiller Universität, Jena, 1999. X+156 oldal, 10 cikk. 6.5

[379] K. Wagner, G. Wechsung: *Computational Complexity*. D. Reidel Publ. Company, 1986. ISBN 9027721467. 6.5

[380] M. A. Weiss: *Data Structures and Algorithm Analysis in C++*. Addison-Wesley, Menlo Park, The Benjamin/Cummings Publ. 1994. 498 oldal, 113 hivatkozás, ISBN 0-8053-5443-3. 6.5

[381] M. A. Weiss: *Data Structures and Algorithm Analysis*. The Benjamin/Cummings Publ. Company, Redwood, 1995. 507 oldal, 408 hivatkozás, ISBN 0-8053-9057-X.

[382] F. Weisz: *Martingale Hardy Spaces and Their Applications in Fourier Analysis*. (*Lecture Notes in Mathematics* **1568**). Springer-Verlag, 1994. 217 oldal, ISBN 038-757-623-1. 6.5

[383] F. Weisz: *Summability of Multi-Dimensional Fourier Series and Hardy Spaces* (*Mathematics and Its Applications* **541**). Kluwer Academic Publishers, Dordrecht, 2002. 352 oldal, ISBN 140-200-564-4. 6.5

[384] H. Wilf: *Generatingfunctionology*. Academic Press, 1990. 224 oldal, 42 hivatkozás. 6.1

[385] H. Wilf: *Algorithms and Complexity*. Második kiadás (az első letölthető a szerző honlapjáról). A. K. Peters, Natick, 2002. 200 oldal, ISBN 1568811780. 6.1

[386] F. Winkler: *Polynomial Algorithms in Computer Algebra*. *Texts and Monographs in Symbolic Computation*. Springer-Verlag, Berlin, 1996, 270 oldal, ISBN 3-211-82759-5. 6.5

[387] J. Winkler: Animations. Friedrich Schiller Universität, Jena, 2003. 6.1, 6.5

[388] N. Wirth: *Algorithms + Data Structures = Programs*. Prentice Hall, Englewood Cliffs, 1976. Magyarul: *Algoritmusok + Adatstruktúrák = Programok* (Fordította: Lehel Jenő). Műszaki Könyvkiadó, Budapest, 1988. 345 oldal, 45 hivatkozás, ISBN 963 103858 0. 6.1

[389] XTANGO: *General purpose algorithm animation system* (szerkeszti John Stasko). Georgia Tech, 2003. 6.1, 6.5

[390] A. Y. Zomaya: *Parallel Computing*. International Thomson Publishing, Cambridge, 1995. 676 oldal, ISBN 1850321884. 6.1

[391] A. Y. Zomaya: *Parallel and Distributed Computing Handbook*. McGraw-Hill, 1995. 1232 oldal, ISBN 0070730202. 6.1

[392] F. Ercal, S. Olariu, A. Y. Zomaya (szerkesztők): *Solutions to Parallel and Distributed Computing Problems: Lessons from Biological Sciences*. John Wiley, 2001. ISBN 0-471-35352-3.

[393] Zsakó László (szerkesztő): *Programozási feladatok. I.* Kossuth Kiadó, Budapest, 1997. 306 oldal, ISBN 963-09-3976-2. 6.5

[394] Zsakó László (szerkesztő): *Programozási feladatok. II.* Kossuth Kiadó, Budapest, 1997. 275 oldal, ISBN 963-09-3977-0. 6.5

Az irodalomjegyzékben az aláhúzás azt jelenti, hogy a DVI és PDF elektronikus változatban a megfelelő szövegrész él (az aláhúzott részre kattintva az olvasó eljut a megfelelő honlapra). A hivatkozások végén lévő számok a szöveg azon részeit jelzik, amelyekben az adott műre hivatkozás történt.

A könyveknél megadjuk az egyedi azonosításra alkalmas ISBN (*International Standard Book Number*) számot [31]. A sorozatokhoz tartozó kiadványoknál (ilyenek a folyóiratok is) az egyedi azonosításra alkalmas, ISSN a (*International Standard Serial Number*) nemzetközi szabványnak megfelelő azonosító számot az L.1. táblázatban adjuk meg.

A *Computing Reviews*, *Compuscience*, *Mathematical Reviews* vagy *Zentralblatt für Mathematik* által referált cikkeknél megadtuk a referátum azonosító adatait.

Lelőhelyjegyzék

Folyóiratok a hazai könyvtárakban

Ebben a részben összefoglaljuk a legfontosabb informatikai folyóiratok nyomtatott és elektronikus változatainak adatait.

A több részből álló első táblázatban – teljes nevük ábécé szerinti sorrendjében – felsoroljuk a legfontosabb informatikai folyóiratok nevét és ISBN azonosítóját. Ennek a listának az alapja a *Fachinformationszentrum* (Karlsruhe) által karbantartott *Compuscience* informatikai adatbázis. A listából kihagytuk a magyar olvasó számára nehezen elérhető folyóiratokat. Ugyanakkor a listát kiegészítettük az összes hazai és az olyan külföldi matematikai folyóiratokkal, amelyek gyakran szerepelnek az általunk idézett szerzők műveiben. A folyóirat neve előtt lévő ++ jel azt mutatja, hogy a folyóirat nem szerepel a *Compuscience* listájában.

A második táblázatban megadjuk a folyóiratok nyomtatott és elektronikus változatainak a 6 legnagyobb hazai informatikai könyvtárban (BME, Debreceni Egyetem, ELTE, Rényi Alfréd Intézet, Szegedi Egyetem, SZTAKI) való elérhetőségét.

A nyomtatott változatok adatai a *Nemzeti Periódika Adatbázis* (NPA) honlapjáról és az NPA által évente kiadott CD-ről származnak. Az elektronikus változatok adataihoz felhasználtuk az említett könyvtárak honlapját.

A folyóiratok nevét az NPA szerint rövidítettük. A nemzetközi rövidítések listája letölthető a referáló folyóiratok honlapjáról.

A táblázatban N a nyomtatott, E az elektronikus változat, K az elektronikus kivonat, T az elektronikus tartalomjegyzék elérhetőségét jelzi (a K jelet használtuk akkor, ha a folyóirat honlapjáról letölthető a cikkek kivonata, és/vagy ha legalább az egyik referáló adatbázis.

Végül az elektronikus adatbázisok, elektronikus könyvtárak, és a konferenciakötetek adatait foglaljuk össze.

Digitális adatbázisok

- Citeseer
- Computing Reviews
- Compuscience
- EFTAN: Egyetemi és Főiskolai Tankönyvek adatbázisa
- ELTE: a Horizon rendszeren keresztül hozzáférés az ELTE, SOTE és a ME könyvtárainak katalógusaihoz
- Kiskapu Kiadó és Könyvesbolt: könyvesboltoknak a magyar nyelvű informatikai szakirodalom teljes választékát kínáló hálózata (rendszeresen közread magyar nyelvű katalógusokat)
- Mathematical Reviews
- Minerva: a Minerva folyóiratokkal kereskedő cég adatbázisa
- NPA: Nemzeti Periodika Adatbázis (az Országos Széchenyi Könyvtár által karbantartott adatbázis a hazai könyvtárakban lévő külföldi és hazai folyóiratokról; 2000-ben abbamaradt az adatok frissítése – viszont az OSZK által évente kiadott NPA-CD friss adatokat tartalamaz)
- Software Station:
- SWETS: a legnagyobb folyóiratkereskedő cég adatbázisa kb. 16000 folyóirat tartalomjegyzékét tartalmazza, emellett a cégen keresztül megrendelt folyóiratok jelentős részének teljes szövegéhez is hozzáférést biztosít
- DLBP Trier

- Universitas: a legjelentősebb magyar cég, amelyen keresztül külföldi kiadású könyvek megrendelhetők. Honlapján sok könyv adatai (például az ára) megtalálhatók.
- Zentralblatt für Mathematik

Digitális könyvtárak

- Academic Press
- ACM Digital Library
- BME-OMIKK:
- EISZ: Elektronikus Információ Szolgálat (ingyenes hozzáférés minden egyetemi hallgató és oktató számára az Elsevier folyóiratainak teljes tartalmához)
- Elsevier
- ELTE (ELTE Informatikai Kara)
- EMIS
- IEEE
- EMIS: European Mathematical Information Service (ingyenes hozzáférés bárki számára 35 matematikai folyóirat teljes tartalmához)
- Kluwer
- MEK (Magyar Elektronikus Könyvtár)
- SIAM
- Springer Verlag
- Typotex

Konferenciák anyagai

L.5. táblázat. Informatikai konferenciák anyagai

Cim	BME	DE	ELTE	RAI	SzTE	SZTAKI
1 *ACM Dist.*	-	E	E	-	E	E
2 *ACM Theory*	-	E	E	-	E	E
3 *ECOOP*	-	E	E	-	E	E
4 *ESA*	-	E	E	-	E	E
5 *FCT*	-	E	E	-	E	E
6 *MFCS*	-	E	E	-	E	E
7 *STACS*	-	E	E	-	E	E

Ebben a táblázatban a *Proceedings of Annual ACM Symposium on Principles of Distributed Computing*, a *Proceedings of Annual ACM Symposium on Theory of Computation*, a *European Conference on Object-Oriented programming*, az *European Sympozium on Algorithms*, a *Foundation of Computing Theory*, a *Mathematical Foundations of Computer Science* és az *Annual Symposium on Theoretical Aspects of Computer Science* nevű konferenciák adatai szerepelnek.

L.1.a táblázat. Folyóiratok nevei és ISBN számai

Sorszám és cím	ISBN
002 *ACM Computing Surveys*	0360-0300
004 *ACM SIGACT News*	0163-5700
027 *ACM Sigplan Notices*	0362-1340
034 *ACM Transactions on Computer Systems*	0734-2071
035 *ACM Transactions on Database Systems*	0362-5915
036 *ACM Transactions on Graphics*	0730-0301
039 *ACM Transactions on Mathematical Software*	0098-3500
041 *ACM Transactions on Programming Lang.*	0164-0925
042 *ACM Transactions on Software Engineering Meth.*	1049-331X
043 *Acta Cybernetica*	0324-721X
044 *Acta Informatica*	0001-5903
++ *Acta Mathematica Academiae Nyíregyháziensis*	0866-0182
++ *Acta Mathematica Hungarica*	0236-5295
++ *Acta Scinetiarum Mathematicarum*	0001-6969
055 *Algorithmica*	0178-4617
056 *Alkalmazott Matematikai Lapok*	0133-3399
058 *The American Mathematical Monthly*	0002-9890
++ *Analysis Mathematica*	0133-3852
++ *Annales Universitatis Eötvös, S. Computatorica*	0138-9491
++ *Annales Universitatis Eötvös, S. Mathematica*	0524-9007
++ *Applied and Computational Harmonic Analysis*	1063-5203
070 *Applied Mathematics and Computation*	0096-3003
073 *Applied Numerical Mathematics*	0168-9274
075 *Ars Combinatoria*	0381-7032
076 *Artificial Intelligence*	0004-3702

L.1.b. táblázat. Folyóiratok nevei és ISBN számai

Sorszám és cím	ISBN
081 *Australasian Journal of Combinatorics*	1034-4942
++ *Automated Software Engineering*	0928-8910
++ *Avtomatika i Telemehanika*	0005-2310
093 *BIT. Numerical Mathematics*	0006-3835
++ *C/C++ User's Journal*	1075-2828
++ *Calculateurs Parallélels*	1260-3198
114 *Combinatorica*	0209-9683
116 *Communications of the ACM*	0001-0782
120 *Complexity*	1076-2787
++ *Computational Complexity*	1016-3328
137 *The Computer Journal*	0010-4620
138 *Computer Languages*	0096-0551
141 *Computer Networks and ISDN Systems*	0169-7552
++ *Computers and Education*	0360-1315
168 *Computers and Operation Research*	0305-0548
173 *Computing. Archiv Inf. Numerik*	0010-485X
++ *Computing Reviews*	0010-4884
178 *Congressus Numerantium*	0384-9684
++ *Current Mathematical Publications*	0361-4794
++ *CWI Quarterly*	0922-5366
189 *Data Knowledge Engineering*	0169-023X
199 *Discrete Applied Mathematics*	0166-218X
201 *Discrete Mathematics*	0012-365X
204 *Distributed Computing*	0178-2770
++ *The Electronic Journal Qualitative Th. Diff. Eq.*	1417-3875

L.1.c. táblázat. Folyóiratok nevei és ISBN számai

Sorszám és cím	ISBN
206 *Dr. Dobb's Journal*	1044-789X
221 *European Journal of Combinatorics*	0195-6698
222 *European Journal of Operation Research*	0377-2217
226 *Formal Aspects of Computing*	0934-5043
229 *Future Generation Computer Systems*	0167-739X
++ *Foundations of Computing and Decision Systems*	0324-8747
230 *Fuzzy Sets and Systems*	0165-0114
232 *Graphical Models*	1524-0703
++ *Higher-Order and Symbolic Computation*	1388-3690
237 *IBM Journal of Research and Development*	0018-8646
238 *IBM Systems Journal*	0018-8670
249 *IEEE Transactions on Communications*	0090-6778
250 *IEEE Transactions on Computers*	0018-9340
252 *IEEE Transactions on Information Theory*	0018-9448
255 *IEEE Transactions on Parallel and Distr. Syst.*	1045-9219
258 *IEEE Transactions on Signal Processing*	1053-587X
259 *IEEE Transactions on Software Engineering*	0098-5589
++ *IEEE Transactions on Visualization Comp. G.*	1077-2626
263 *IMA Journal on Numerical Analysis*	0272-4979
272 *Information and Computation*	0890-5401
273 *Information and Control*	0019-9958
279 *Information Processing Letters*	0020-0190
++ *Java Developer's Jounal*	1087-6944
327 *Journal of Algorithms*	0196-6774
331 *Journal of Automated Reasoning*	0168-7433

L.1.d. táblázat

Cím	Név
334 *Journal of Combinatorial Theory, Series A*	0097-3165
335 *Journal of Combinatorial Theory, Series B*	0097-3165
337 *Journal of Computational and Appl. Math.*	0377-0427
341 *Journal of Computer and Systems Sci.*	0022-0000
++ *The Journal of Fourier Analysis and Applications*	1069-5869
++ *Journal of Functional Programming*	0956-7968
++ *Journal of Grid Computing*	indul
++ *Journal of Graph Algorithms and Applications*	1526-1719
356 *Journal of Graph Theory*	0364-9024
++ *Journal of Integer Sequences*	1530-7638
++ *Journal of Logic, Language and Information*	0925-8531
364 *Journal of Logic Programming*	
377 *Journal of Parallel and Distributed Computing*	0743-7316
++ *Journal of Systems and Software*	0164-1212
390 *Journal of the ACM*	0004-5411
++ *Lecture Notes in Computer Science*	0302-9743
417 *Management Sciences*	0025-1909
++ *Mathematica Pannonica*	0865-2090
++ *Mathematical Methods in Applied Sciences*	0170-4214
430 *Mathematics of Computattion*	0025-5718
454 *Neural Networks*	0893-6080
++ *Nieuw Arakief voor Wiskunde*	0028-9825
464 *Numerische Mathematik*	0029-599X

L.1.e. táblázat

Cím	Név
++ *L'Objet*	1262-1137
++ *Operating Systems Review*	0163-5980
468 *Operations Research*	0030-364X
++ *Operation Research Letters*	0167-6377
472 *Parallel Computing*	0167-8091
479 *Performance Evaluation*	0166-5316
480 *Periodica Mathematica Hungarica*	0031-5303
490 *Proceedings of IEEE*	0018-9219
495 *Programmirovanie*	032-3474
++ *Publicationes Mathematicae*	0033-3883
++ *Pure Mathematics and Applications*	1218-4586
++ *Quarterly of Applied Mathematics*	0033-569X
499 *RAIRO. Informatique Theoretique et Applications*	0988-3754
502 *Random Structures and Algorithms*	1042-9832
515 *Science of Computer Programming*	0167-6423
517 *SIAM Journal on Applied Mathematics*	0036-1399
518 *SIAM Journal on Computing*	0097-5397
519 *SIAM Journal on Control and Optimization*	0363-0129
520 *SIAM Journal on Discrete Mathematics*	0895-4801
523 *SIAM Journal on Numerical Analysis*	0036-1429
524 *SIAM Journal on Optimization*	1052-6234
526 *SIAM Journal on Scientific Computing*	1064-8275
527 *SIAM Review*	0036-1445

L.1.f. táblázat

Cím	Név
532 *Simulation*	0037-5497
++ *Sysadmin and Perl Journal*	1061-2688
540 *Software Practice and Experience*	0038-0644
++ *Studia Scientiarum Mathematicarum*	0081-6906
++ *Studia Universitatis Babes-Bolyai, Informatica*	1224-869x
561 *Theoretical Computer Science*	0304-3975
573 *VLDB Journal*	1066-8888
++ *Zentralblatt für Mathematik*	1436-3356

L.2.a. táblázat. Informatikai folyóiratok a magyar könyvtárakban

Sorszám és cím	BM	DE	EL	RA	SE	SZ
002 *ACM Surveys*	E	E	NE	-	NE	NE
004 *ACM SIGACT News*	E	E	E	E	E	
027 *ACM Sigplan Notices*	E	E	E	E	E	
034 *ACM Trans. Comput. Syst.*	NE	E	NE	E	E	E
035 *ACM Trans. Database Syst.*	NE	E	NE	E	E	E
036 *ACM Trans. Graph.*	E	E	E	E	E	
039 *ACM Trans. Math. Softw.*	E	E	E	E	E	NE
041 *ACM Trans. Program. Lang.*	NE	E	NE	NE		NE
043 *Acta Cybernetica*	K	K	NK	K	NK	NK
044 *Acta Inf.*	E	E	NE	E	E	NE
++ *ActaNyíregyháziensis*	E	E	E	E	E	E
++ *Acta Mathematica Sci.*		N	N		N	
055 *Algorithmica*	E	E	NE	E	E	NE
055 *Algorithmica*	E	E	NE	E	E	E
056 *Alkalmazott Matematikai Lapok*	NK	NK	NK	NK	NK	NK
058 *The American Math. Monthly*	E	E	NE	E	E	E
++ *Analysis Mathematica*	T	T	NT	T	T	
++ *Ann. Univ. Eötvös, Computatorica*	NT	NT	NT	NT	NT	NT
++ *Ann. Univ. Eötvös, Mathematica*	N	N	N	N		
++ *Applied Comp. Harmonic An.*	E	E	NE	E	E	E
070 *Applied Math. Comput.*			N			
073 *Applied Numerical Math.*	N					
075 *Ars Combinatoria*	K	K	K	K	N	
076 *Artif. Intell.*	E	E	NE	E	E	NE
081 *Australasien Journal Comb.*	K	K	K	K	K	
++ *Automated Software Engineering*			N			

L.2.b. táblázat. Informatikai folyóiratok a magyar könyvtárakban

Sorszám és cim	BM	DE	EL	RA	SE	SZ
++ *Avtomatika i Telemehanika*			N			
093 *BIT. Numerical Mathematics*			NE			
++ C/C ++ *User's Journal*			N			
++ *Calculateurs Parallélels*			N			
114 *Combinatorica*	NE	NE	NE	NE	NE	NE
116 *Communications of ACM*	NE	E	NE	NE	E	NE
120 *Complexity*	E	E	E	E	E	
++ *Computational Complexity*	E	E	E	E	E	NE
137 *The Computer Journal*			NE			NE
138 *Computer Languages*			N			
141 *Computer Networks ISDN Syst.*	E	E	EN	E	E	E
++ *Computers and Education*			N			
168 *Computers and Op. Research*	E	E	NE	E	E	E
173 *Computing. Archiv für Informatik*	E	E	N	E	E	
++ *Computing Reviews*	E	E	NE	NE	E	
178 *Congressus Numerantium*				N		
++ *Current Mathematical Publications*			N			
++ *CWI Quarterly*	E	E	EN	E	E	
189 *Data Knowledge Engineering*			N			
199 *Discrete Applied Mathematics*	E	E	NE	NE	E	E
201 *Discrete Mathematics*	E	E	NE	NE	E	E
204 *Distributed Computing*	NE	E	NE	E	E	
206 *Dr. Dobb's Journal*	E	E	NE	E	E	E
++ *Electronic Journal of Comb.*	E	E	E	E	E	E
++ *Electronic Num. Anal.*	E	E	E	E	E	E

L.2.c. táblázat. Informatikai folyóiratok a magyar könyvtárakban

Sorszám és cim	BM	DE	EL	RA	SE	SZ
221 *European Journal of Comb.*	E	E	E	E	E	E
222 *European Journal Op. Research*	E	E	NE	E	E	E
226 *Formal Aspects of Computing*	E	E	NE	E	E	E
++ *Foundations of Computing Dec.*			N			
++ *Fundamenta Mathematicae*			T	N		
229 *Future Generation Comp. Syst.*			T	N		
230 *Fuzzy Sets and Systems*	E	E	E	E	E	
232 *Graphical Models*	E	E	NE	E	E	
++ *Higher-Order and Symb. Comp.*	E	E	NE	E	E	
237 *IBM Journal of Research Dev.*	E	E	NE	E	E	
238 *IBM Systems Journal*	E	E	NE	E	E	
249 *IEEE Transactions Comm.*			NK			NE
249 *IEEE Trans. Comput.*	NE	NK	NK	K	K	K
250 *IEEE Transactions Inf. Th.*	NK	K	NK	K	K	NK
259 *IEEE Transactions on Software*			EN			EN
++ *IEEE Transactions Visualization*	E	E	NE	E	E	NE
263 *IMA Journal Num. Analysis*	E	E	NE	E	E	
272 *Information and Computation*			KN			EN
273 *Information and Control*			TK	N		
279 *Information Processing Letters*	E	E	NE	E	NE	NE
++ *Java Developer's Journal*	E	E	E	E	E	E
327 *Journal of Algorithms*			KN			E

L.2.d. táblázat. Informatikai folyóiratok a magyar könyvtárakban

Cim	BM	DE	EL	RA	SE	SZ
334 *Journal of Comb. Theory, A*	TK	TK	NK	TK	TK	TK
335 *Journal of Comb. Theory, B*	TK	TK	NK	TK	TK	TK
++ *Journal of Computer Syst.*			EN			EN
++ *Journal of Graph Alg.*	E	E	E	E	E	E
++ *Journal of Grid Computing*						
++ *Journal of Integer Sequences*	E	E	E	E	E	E
390 *Journal of ACM*		E	NE		E	
++ *Lecture Notes in CS*	E	E	EN	E	E	
427. *Math. Syst. Theory*			N			
++ *Nieuw Arakief*			N			
++ *Nordic J. Com.*			IT			
490. *Proc. IEEE*	NK	K	NK	K	K	NK
++ *Publ. Math.*	NT	NT	NT	NT	NT	NT
++ *Q. Appl. Math.*		N			N	E
518. *SIAM J. Comp.*			NE		NE	E
520. *SIAM Discret.*			K	KN	EN	EN
527. *SIAM Rev.*	NE	K	NE	K	K	NE
561. *Theor. Comput.*	E	E	EN	E	E	E
++ *Zentralbl. Did. der Math.*	E	E	E	E	E	E

Névmutató

A névmutatóban a magyar szerzők utóneveit is teljesen kiírtuk, míg a külföldi szerzők esetében – mivel a teljes nevet nem minden esetben tudjuk – rövidítettük az utóneveket.

A, Á

Abello, J., 237, 253
Adriaans, P., 253
Aho, A. V., 225, 253
Akl, S., 228, 253
Almasi, G. S., 226, 227, 253
Álmos, Attila, 254
Amdahl, G. M., 36, 70, 227, 254
Amestoy, P., 227, 235, 254
Andrásfai, Béla, 227, 230, 254
Arató, Péter, 235, 254
Argyros, I. K., 238, 254
Artiaga, L., 225, 254
Atallah, M. J., 254
Ausiello, G., 234, 255
†Aven, O. I., 234, 255

B

Baase, S., 226, 255
Bach, Iván, 235, 255
Baeza-Yates, R., 264
Bagyinszki, János, 255, 263
Bahill, A. T., 238, 286
Bahvalov, N. N., 238, 255
Bajalinov, Erik, 255
Baksa, Klára, 13

Balázs, Gábor, 13
Balogh, Ádám, 13
Barnes, G. H., 255
Bélády, László, 62
Belényesi, Viktor, 14, 229, 255
Bellman, R. E., 116
Benczúr, András, 262, 288
Bent, D. H., 229, 277
Berge, C., 230, 256
Berke, Barnabásné, 256
Berman, K. A., 225, 228, 230, 256
Bernoulli, J., 51, 87
Bodlaender, H. L., 231, 256
Bollobás, Béla, 228, 230, 256
Boros, László, 256
Boxer, L., 226, 277
Brassard, G., 226, 256
Bratley, P., 226, 256
Brauer, W., 283
Brent, R. P., 36
Bronstejn, I. N., 256
Brown, C. A., 228, 281
Brucker, P., 236, 256
Burks, W., 278
Busacker, R. G., 230, 256

C

Carlsson, C., 257
Censor, Y., 257
Chandy, K. M., 238, 257
Chang, E., 206, 231, 257
Chow, R, 257
Chua, L., 236, 257
Coffman, E. G. jr., 228, 229, 234, 255, 257
Cole, R., 258
Cormen, T. H., 225, 228, 237, 258
Crescenzi, P., 255

CS

Cseke, Vilmos, 258
Csernov, H., 51, 54, 87, 132, 177, 178, 228, 257
Cserny, László, 258
Csirik, János, 229, 234, 257, 258, 287
Csizmazia, Balázs, 259
Csörnyei, Zoltán, 235, 236, 259

D

Davis, L., 225, 254

De Berg, M., 256
de Bruijn, N. G., 42
Demetrovics, János, 227, 259
Dénes, József, 259
Denev, J., 227, 259
Dévai, Gergely, 13
Devroye, L., 259
Diestel, R., 230, 259
Dolev, D., 231, 259
Donovan, J. J., 236, 259
Dossey, J. A., 233, 259
Dózsa, Gábor, 13
Dringó, László, 227, 259
Drommerné, Takács Viola, 226, 236, 259
Du, D.-Z., 260
Dulff, S., 229, 270
Dynes, C., 237, 263

E, É

Ercal, F., 290
†Erdős, Pál, 228, 233, 260
Ésik, Zoltán, 279
†Euler, L., 196
Eynden, C. V., 233, 259

F

Fábián, Mária, 13
Farin, G. E., 260
Farkas, Zsuzsa, 234, 260
Fehérvári, Arnold, 225, 276
Fekete, István, 228, 237, 260, 262, 267
Feller, W., 228, 260
Ferenczi, S., 229, 260
Fiat, A., 260
Flach, P., 233, 261
Flajolet, P., 228, 284
Floudas, C., 261
Floudas, C. A., 238
Floyd, R. W., 230, 261
Flynn, M., 31
Flynn, M. J., 227, 261
Ford, L., 116
Forgó, F., 261
Fornai, Péter, 63, 228, 261
Fortune, S., 261
Fóthi, Ákos, 4, 235, 261
Fountain, T., 227, 284
Fourier, J. B. J., 160
Frank, András, 238, 261
French, S., 236, 261

Frenk, J. B. G., 229, 258
Freud, Róbert, 233, 261
†Frey, Tamás, 226, 262
Fullér, Róbert, 257, 262, 273
Futó, Iván, 234, 260, 262
Fülöp, Zoltán, 235, 262
Füsi, János, 262

G

Gács, Péter, 225, 226
Gagne, G., 284
Galambos, Gábor, 229, 236, 258, 262
Galántai, Aurél, 262
Galvin, P., 284
Gambosi, G., 255
Garcia-Molina, H., 262
Garey, M. R., 227, 234, 263
Gavrilov, G. P., 263
Gergó, Lajos, 233, 238, 263, 277
Gibbons, A., 226, 228, 263
Gilchrist, B., 263
Girault, C., 236, 263
Gloor, P., 237, 263
Goldberg, D. E., 263
Gonda, János, 227, 263, 274
Gonnet, G. H., 225, 264
Gottlieb, A., 226, 227, 253
Graham, R. L., 226, 228, 271
Gray, F., 169, 196
Green, D. H., 264
Greenflaw, R., 226, 264
Gregorics, Tibor, 237, 260, 262
Gruska, J., 225, 263, 264
Gustafson, J., 37, 70, 227, 264

GY

Gyarmati, Edit, 233, 262
Gyimóthy, Tibor, 237, 262
Györfi, László, 259, 264
György Anna, 255
Győri, Sándor, 254, 264

H

Halmos, P. R., 228
†Hamilton, W. R., 196
Hamming, R. W., 164
Harris, T. J., 264
Hatvani, László, 264
Haupt, R. L., 265
Haupt, S. E., 265

Heath, M. T., 228, 265
Hecker, H.-D., 228, 265
Hegyessy, Tamás, 13
Hemmendinger, D., 225, 282
Hennessy, J. L., 235, 265
Henrici, R., 238, 265
Hermann, Péter, 14
Hernyák, Zoltán, 266
Hirschberg, D. S., 208, 265
Hochbaum, D., 234, 265
Hofri, M., 228, 265
Hoover, J., 226, 264
Hopcroft, J. E., 225, 235, 253, 265
Horowitz, E., 227, 230, 265
Horváth, Arnold, 233, 277
Horváth, Gábor, 254
Horváth, László, 226, 235, 266
Horváth, Márk, 266
Horváth, Z., 266
Horváth, Zoltán, 4, 13, 261, 266
Hujter, Mihály, 262
Hunyadvári, László, 228, 235, 267
Hwang, K., 235, 267

I, Í

Imreh, Balázs, 229, 255, 258, 264, 267
Iványi, A., 267
Iványi, Anna, 14, 235, 267
Iványi, Antal, 3, 4, 63, 228, 229, 234, 236, 266, 268, 276
Iványi, Antalné, 268
Ivanyos, Gábor, 225, 282

J

Jájá, J., 226, 228, 268
Jakobi, Gyula, 235, 269
Jankovits, István, 235, 254
Járai, Antal, 228, 269
Jeffres, L. A., 269
Jenei, András, 262
Johnson, D. S., 227, 229, 234, 258, 263
Johnson, T., 257
Jones, N. D., 269
Juhász, István, 285
Jungnickel, D., 230, 269
Jutasi, István, 236, 269

K

Kacsuk, Péter, 227, 235, 269, 284
Kallós, Gábor, 233, 277
Kalmár, Zsolt, 236, 276
Kapinya, Judit, 13
Kása, Zoltán, 4, 13, 225, 229, 230, 260, 269
Kátai, Imre, 227, 229, 234, 259, 268, 269, 271
Katona, Gyula, 274
Katona, Y. Gyula, 228, 230, 270
Kaufman, A., 270
Keedwell, A. D., 259
Kelemen, József, 237, 270
Kemnitz, A., 229, 270
Kernighan, B. W., 270
Kingston, J. H., 270
Kiss, Attila, 282
Kiss, Béla, 270
Kiss, Ottó, 270, 282
Klawe, M., 231, 259
Klein, Sándor, 270
Klincsik, Mihály, 233, 271
Knuth, D. E., 225, 226, 264, 271
Kobelkov, G. M., 238
Kogan, Y. A., 234, 255
Kohler, M., 264
Komlósi, Sándor, 238, 272
Korach, E., 231, 272, 279
Kormos, János, 285
Korolev, L. N., 272
Kósa, András, 264
Kotov, V. E., 236, 272
Kotsis, G., 269
Kotsis, Gabrielle, 227
Kovács, Attila, 272
Kovács, Előd, 272
Kovács, Gábor Zsolt, 229, 272
Kovács, Győző, 272
Kovács, Margit, 239, 255, 270, 273
Kovács, Péter, 13
Kovács, Zoltán, 235, 268
Kozics, Sándor, 273
Kozma, László, 238, 273
Kozsik, Tamás, 261, 266, 273
Kőhegyi, János, 235, 273
Kranzlmüller, D., 227, 235, 269
Krebsz, Anna, 270
Kruskal, J. B., 115, 116
Krzysak, A., 264
Kurtán, Lajos, 274

L

Lai, T., 68
Lai, T. H., 228, 274
Láng, Csabáné, 227, 274
Langdon, W. B., 274
Langer, Tamás, 234, 260
Lavrov, I. A., 274
Lawler, E. L., 225, 274
Lee, C., 228, 258
Lee, I., 237
Lee, L., 263
Lehel, Jenő, 289
Lehmer, D. H., 274
Leighton, T. F., 227, 230, 231, 275
Leiserson, C. E., 225, 228, 237, 258
LeLann, G., 202, 231, 275
Leopold, C., 228, 275
Levitin, A. A., 225, 275
Li, M., 234, 275
Lin, E., 228, 258
Locher, Kornél, 14
Lovász, László, 225, 226, 233, 234, 275
Lőcs, Gyula, 275
Lőrincz, András, 236, 276
†Lucas, É., 276
Lugosi, Gábor, 259
Lynch, N. A., 201, 226–228, 231, 232, 276

M

Malyshkin, V., 276
Manherz, Tamás, 235, 267
Maple, 233
Marchetti-Spaccamela, A., 255
Marczell, Zsolt, 236, 276
Maróti, György, 233, 271
Martello, S., 229, 257
Marton, László, 225, 276
Mathematica, 233
Matlab, 233
Maximova, L. L., 274
Mayr, E. W., 228, 276
McCarthy, J., 284
Mehlhorn, K., 276
Metykó, Beáta, 14
Miletics, Edit, 13
Miller, R., 226, 277
Misra, J., 238, 257, 277
Mitchell, M., 277

Molnár, Ervin, 270
Molnárka, Győző, 233, 277
Moon, J. W., 277
Moran, S., 272
Morgenstern, O., 277
Motwani, R., 228, 235, 265, 277

N

Nagy, Sára, 237, 260, 270
Nagy, Tibor, 237, 277
Narayana, T. V., 229, 277
Nassimi, D., 277
Nelson, R. A., 62
Németh, Cs. Dávid, 229, 255
Németh, Zsolt, 227, 235, 269
†Neumann, János, 31, 226, 277, 278
Nijenhuis, A., 233, 278
Norton, P., 236, 278
Norvig, P., 237, 283

NY

Nyékyné, Gaizler Judit, 235, 278

O, Ó

Obádovics, J. Gyula, 238, 279
Okuguchi, K., 279
Olariu, S., 290
Omega, 243
Ore, O., 230, 279
Otto, A. D., 233, 259
Overmas, M, 256

P

Pachl, J., 231, 279
Pál, Jenő, 239, 284
Pándi András, 4
Papadimitriou, C. H., 225, 234, 279
Pardalos, P. M., 228, 237, 238, 253, 260, 261, 280
Parent, R., 280
Pásztorné, Varga Katalin, 233, 280
Pataki, Norbert, 229, 272
Patashnik, O., 226, 271
Patterson, D. A., 235, 265
Paul, J. L., 225, 228, 230, 256
Pavlov, R., 227, 259
Pécsy, Gábor, 4, 13, 229, 280
Pergel, József, 229, 268

Peterson, G. L., 231, 280
Pethő, Attila, 13, 233, 280
Pike, R., 270
Pirkó, József, 226, 235, 266, 280
Poli, R., 274
†Pongor, György, 235, 280
Porkoláb, Zoltán, 286
Prather, R. E., 233, 281
Prékopa, András, 228, 281
Preparata, F. P., 108, 230, 281
Prim, R. C., 116
Protasi, M., 255
Prömel, H. J., 228, 276
Purdom, P. W., 228, 281
Pyle, I. C., 281

Q

Quinn, M. J., 281

R

Raghavan, P., 228, 277
Rajasekaran, S., 228, 230, 266, 280, 281
Ralston, A., 225, 281, 282
Ranade, A., 228, 265
Ranka, S., 231
Recski, András, 270, 282
Reif, J. H., 282
Reilly, E. D., 225, 282
Reischuk, R., 110, 230, 282
†Rényi, Alfréd, 228, 282
Resende, M. G. C., 237, 238, 253, 280
Rét, Anna, 13
Rinnoy Kan, A. H. G., 229, 258
Ritchie, D. M., 270
Rivest, R. L., 225, 228, 230, 237, 258, 261
Roberts, R., 206, 231, 257
Rodeh, M., 231, 259
Rodríquez-Vázquez, A., 283
Rolland, F., 282
Rónyai, Lajos, 225, 282
Roosta, S., 227, 228, 283
Roska, Tamás, 236, 257, 283
Rotem, D., 231, 279
Rozenberg, G., 283
Russell, S. J., 237, 283
Ruzzo, J. W., 226, 264
Ruzsa, Imre, 233, 283
Rytter, W., 226, 228, 263, 283

S

Saaty, T. L., 230, 256
Sack, J.-R., 230, 283
Sahni, S., 68, 228, 230, 231, 266, 274, 277
Salamon, Gábor, 237, 283
Salleh, S., 283
Salomaa, A., 283
Sántáné, Tóth Edit, 237, 262, 284
Schipp, Ferenc, 284, 285
Schöning, U., 284
Schreiber, R. S., 228, 265
Schreiner, W., 228, 285
Schwarzkopf, O., 256
Sedgewick, R., 228, 284
Selim, G. A., 226, 284
Sen, S., 281
†Shannon, C. E., 284
Shedler, G. S., 62
Sike, Sándor, 238, 288
Siklósi, Bence, 229, 284
Silberschatz, A., 284
Sima, Dezső, 13, 227, 284
Simon, Péter, 238, 239, 284, 285
Simonovics, Miklós, 271
Sinclair, J. B., 208, 265
Sipos, Annamária, 285
Sipser, M., 234, 285
Skiena, S. S., 226, 285
Sommerville, I., 238, 285
Spence, L. E., 233, 259
Spencer, J., 228, 260, 285
Stanley, R. P., 285
Stavros, A. Z., 257
Steiger, A., 228, 276
Steiglitz, K., 279
Stein, C., 225, 228, 258
Steingart, Ferenc, 235, 261
Stirling, J., 68
Stockman, M., 236, 278
Stoyan, Gisbert, 238, 285
Stroustrup, B., 286
Sussman, J., 228

SZ

Szabó, Csaba, 230, 270
Szabó, Réka, 225, 282
Szadovszkij, A. L., 229, 283
Szadovszkij, L. E., 229, 283
Szalai, Róbert, 14
Szántai, Tamás, 238, 239, 286
Szapozsenko, A. A., 263

†Száva, Géza, 285
Szeberényi, Imre, 238, 286
Szelezsán, János, 226, 262
Szemengyajev, I. N., 256
Szép, J., 261
Szepesváry, Csaba, 236, 276
Szeredi, Péter, 234, 260
Szidarovszky, Ferenc, 238, 254, 279, 286
Szili, László, 13, 233, 286
Szirmay-Kalos, László, 237, 286
Szlávi, Péter, 235, 287
Szmeljánszkij, R. L., 228, 236, 268
Szűcs, László, 4, 13, 229, 280

T

Takó, Galina, 285
Tanenbaum, A. S., 231, 236, 287
Tatai, Gábor, 237, 262
Tejfel, Máté, 266
Tel, G., 201, 227, 231, 232, 287
Tóth, János, 233, 286
Tóth, Zoltán, 267
Trahtenbrot, B. A., 225, 287
Tsantilas, T., 281
Turcsányiné, Szabó Márta, 287

U, Ú

Ulbert, Attila, 266
Ullman, J. D., 225, 253, 262, 265, 288
Urbán, János, 274
Urrutia, J., 230, 283

V

Vajda, István, 264
Valiant, L., 282
Valiente, G., 225, 226, 288
Valk, R., 236, 263
Van Gelder, A., 226, 255
Van Kreveld, M., 256
Van Leeuwen, J., 288
Van Steen, M., 231, 287
Varga, László, 236–238, 273, 288
Várkonyiné, Kóczy Attila, 254
Várterész, Magdolna, 233, 280
Vasziljev, F. P., 273
Vazirani, V. V., 234, 288
Venczel, Tibor, 266
Vida, János, 288

Vigassy, József, 276
Vigo, D., 229, 257
Vilenkin, N., 233, 288
Visegrády, Tamás, 235, 254
Vishkin, U., 258
Vitanyi, P., 234, 275
Vizvári, Béla, 236, 239, 288
Vogel, J., 288
Volkert, J., 227, 235, 269

W

Wade, W. R., 239, 284
Wagner, K., 288
Walk, H., 264
Wechsung, G., 288
Weiss, M. A., 289
Weisz, Ferenc, 239, 289
Wettl, Ferenc, 233, 277
Widom, J., 262, 288
Wilf, H., 226, 228, 233, 278, 289
Willie, J., 261
Winkler, F., 233, 289
Winkler, J., 228, 237, 289
Winkler, Zoltán, 272
Wirth, N., 225
Woeginger, G. J., 229, 234, 257, 258, 260
Woodhull, A. S., 236, 287

Y

Yakowitz, S., 238, 286

Z

Zajki, László, 281
Zaks, S., 272
Zantinge, D., 253
Zhidkov, E. P., 255
Zomaya, A. Y., 228, 283, 290

ZS

Zsakó, László, 235, 287, 290
Zsidkov, E. P., 238

Tárgymutató

Ez a tárgymutató a következő szempontok szerint készült.

A számokat és görög betűket tartalmazó tárgyszavakat kiejtésük szerint rendezzük: például az „1-értékű”-t „egyértékékű”-ként, a λ-t „lambda”-ként. A jelölést tartalmazó tárgyszavakat elemeik szerint rendezzük: például a „k-megegyezés”-t „k megegyezés”-ként.

A különböző típusú objektumokat lehetőség szerint tipográfiailag is megkülönböztettük. A matematikai jelöléseket és a programokban használt változók neveit dőlt betűk emelik ki, mint például $\Omega(n \lg n)$ vagy *rang*[*lépés*]. Az algoritmusok neveit kis kapitális betűkkel írtuk, mint például K ´-
. Az algoritmusok kódjában a programozási alapszavakat félkövéren szedtük, mint például **if, then, for, in parallel for**.

Az algoritmusok nevében kiskötőjelet használtunk, viszont a változók neveiben alsó kötőjelet, mint például P´ - és *bal_szomsz.* Az egyes fogalmak meghatározásának helyére a tárgymutató dőlt oldalszámmal utal.

Elsősorban az algoritmusokat tárgyaló tankönyvek matematikai jelöléseit alkalmaztuk. Az oldalszámok felsorolásánál nem törekedtünk teljességre.

A, Á

abszolút nagy ordó, 242
abszolút optimális algoritmus, *26*
absztrakt számítógép, *lásd* számítási modell
ACM, 267
adatfeldolgozás
  párhuzamos, 15
  soros, 15
adatkígyó, *146*
adatkígyók rendezése, 147
adat koncentráció, 179
adatkoncentráció, 132, *137*
  hiperkockán, 179
algoritmus
  abszolút optimális, 26
  aszimptotikusan optimális, 25
  Las Vegas, 50
  mohó, 129
  Monte Carlo, 50
  munkahatékony, 25
  neurális hálókban, 236
  párhuzamos, 19
  rekurzív, 45
  soros, 19
  tervezése, 15
algoritmusok
  párhuzamos, 225
  soros, 225
általános csomagirányítási feladat, 128
Amdahl törvénye, 36
animáció, 237
anomália, *61*, 70
asszociatív művelet, 73
aszimptotikusan azonos nagyságrend, 26
aszimptotikusan optimális algoritmus, *25*
aszinkron processzorok, 16
átfedő csomagok, *176*
Á , *47*
átmérő, 132, 179
azonosító, 202

B

beágyazás, 167
  fáé, 171
  gyűrűé, 168
  tóruszé, 170
beágyazás késleltetése, *167*
beágyazás torlódása, 167

beemelés, *84*
bejárási algoritmus, 196
Bellman–Ford-algoritmus, 116
Bernoulli-kísérlet, 51, 87
bináris fa, 161
bináris fa alakú hálózat, 171
bináris fa hálózat, *38*
bitfordító gyűrű, 216
biton rendezés, 193
biton sorozat, *191*
blokkonként kígyószerű sorfolytonos indexelés, 135

C

c-szimmetrikus gyűrű, 216
CALGO, 257
C -R , 205, *206*, 215
CRCW, *32*, 91, *lásd* párhuzamos olvasás – párhuzamos írás
CRCW PRAM, 115, 116
CREW, *32*, *lásd* párhuzamos olvasás – kizárólagos írás
CREW PRAM, 34, 74, *lásd* párhuzamos gép

CS

Csernov-egyenlőtlenség, 177, 178
Csernov-egyenlőtlenségek, 51, 132
csillag, 38
csomagirányítás
  közeli célokokhoz, 160
  rácson, 160
  tóruszon, 160

D

d-reguláris, 164
de Bruijn-hálózat, *42*
D - ´ - ´ , 142
D - , 81
diszjunkció, *lásd* logikai összeadás
döntési fa, *58*

E, É

E ´ - ´ , *91*
egy célcsomagos feladat, *126*
egy kezdőcsomagos feladat, 125
egyszerű algoritmus, 100
elem rangja, 79
ellenfél,

elsőbbségi szabály, 122

él torlódása, *167*

ERCW, *32*, *lásd* kizárólagos olvasás – párhuzamos írás

ERCW PRAM, 35

EREW, *32*, 122, *lásd* kizárólagos olvasás – kizárólagos írás

EREW PRAM, 35, 75, 115

EREW-prefix, 74, *76*

érintő, 155

értesítő üzenet, 205

érvényességi feltétel, 219

Euler-kör, 196

exponenciális lépésszám, *21*, *22*, *lásd* szuperpolinomiális lépésszám

F

fa beágyazása, 171

fa magassága, 161

FDD, 126

FDF, *122*, 158

feladat megfogalmazása, 15

felezési szám, *44*

felfúvódás, *167*, 170

FFT, 160, *lásd* gyors Fourier-transzformált

FIFO, 60, *122*, 123, 158

FOF, *122*, 158

G

gombócevési sebesség, 64

gráfalgoritmusok, 151

Gray-kód, 169, 196

Gustafson törvénye, 37

GY

gyenge nagy ordó, 242

gyenge polinomiális lépésszám, *21*

gyors Fourier-transzformált, 160

G ¨ ¨ - , *90*

gyűrű, 39, 168

gyűrű hálózat, 216

H

H - , 220

hálózat

  bináris fa, 38, 171

  csillag, 38

  de Bruijn, 42

háromdimenziós rács, 40
henger alakú, 40
hiperkocka, 42, 163
keverő-cserélő, 199
permutációs, 42
pillangó, 42
piramis, 41
tégla alakú, 40
tórusz, 41
hálózatok, 226
Hamilton-kör, 196
Hamming-távolság, 164
H - , *47*, 70
H´ - ´ ´, 131, 160, 183
háromdimenziós rács, *40*
háromszögmátrix invertálása, 162
H ´ -¨ ´ ¨ , *102*
H ´ - , 74
hatékonyság, 116
hatékonysági mérték, *24*
abszolút, 22
relatív, 22
henger, 40
hibavalószínűség, *50*
hibrid processzorok, 16
hiperkocka, *42*, 163, 186, 192
párhuzamos, 164
soros, 164
hiperkocka fokszáma, 164
hiperkocka i-edik sora, 184
H -S , 215, 216

I, Í

i-edik sor, *184*
I ˝- , 208, 221, 222
időzítés, 16
i-edik szintű kapcsolat, 164
IEEE, 267
ILLIAC-IV, 31, 40
inverzió, 58
ISBN, 290, *lásd* International Standard Book Number
I ´ - , *52*
ISSN, 290, *lásd* International Standard Serial Number

J

j-edik oszlop, *184*

K

k-dimenziós rács, *39*, *119*

k-megegyezés, 212
k-adrendű Gray-kód, *169*
keresés, *88*
kereszt kapcsolat, *165*
késleltetés, 170
K´ - ´ , *129*
kétfázisú algoritmus, 129
K - ´ , 184
kezdő processzor, *202*
K , *84*
kiemelés, *84*
kígyószerű sorfolytonos indexelés, 135
kis omega, *20*
kis ordó, *20*, 242
kiválasztás, *87*, 139, 187
 hiperkockán, 186, 187
 rácson, 139
k-*k* irányítási feladat, *132*
kocka, 17, *40*, 119, *121*, 151, 152
K - ´ , *153*
kommunikációs bonyolultság, 222
kommunikációs hiba,
kommunikációs vonal, 39, 119
konvex burok, 155, 159, 196, 230
konvex burok területe, 162
korlátozás és szétválasztás, 66
korlátozó függvény, 67
köbös lépésszám, *21*
körmentes gráf, 117
körmentesség, 161
közös EREW PRAM, 35, *lásd* párhuzamos gép
közvetlen hozzáférésű gép, 31
közvetlen kapcsolat, *165*
Kruskal-algoritmus, 115, 116
kulcsok rangja, 145
kupac, 67

L

lánc, *39*, *119*
lapozási sebesség, *61*
Las Vegas algoritmusok, *50*
LEDA, 274, 277
legnagyobb helyi értékű bitek, 171
legrövidebb út, 155
legrövidebb utak, 195
legtávolabbra utazó csomag először, 122
legtávolabbról jött csomag

először), 122
L L , *204*, 215
lépésszám, 123
  átlagos, 22
  csomagirányítási algoritmusé, 123
  exponenciális, 21
  gyenge polinomiális, 21
  konstans, 21
  köbös, 21
  legjobb esetben, 22
  legrosszabb esetben, 22
  lineáris, 21
  logaritmikus, 21
  majdnem konstans, 21
  négyzetes, 21
  polilogaritmikus, 21
  polinomiális, 21
  szublineáris, 21
  szublogaritmikus, 21
  szubpolinomiális, 21
  szuperlineáris, 21
  szuperlogaritmikus, 21
  szuperpolinomiális, 21
  várható, 22
levél szintje, 59
LIFO, 158
lineáris lépésszám, *21*
L -¨ ´ ¨ , 97
logaritmikus lépésszám, *21*
L -¨ , *35*
logikai összeadás, 35
(l + 1)-edik szintű kapcsolat, *165*

M

Markov-egyenlőtlenség, 51
mátrix invertálása, 162
  háromátlósé, 162
  háromszögmátrixé, 162
M , *57*
M - , 217
maximális relatív lépésszám, 36
megbízhatóság, 16
megegyezési feltétel, 219
megengedett megoldás, 66
meghibásodás, 226
megoldási fa, 66
megoldhatatlan feladat, 201
megoldhatatlanság, 201
MIMD, *31*
minimális feszítőfa, 161
minimális költségű csúcs, 66

minmátrix, 152, 193
MISD, *31*
mohó algoritmus, 129
Monte Carlo algoritmus, *50*
MSB, *lásd* legnagyobb helyi értékű bitek
munka, *24*, 116
munkahatékony algoritmus, *25*
mutatóugrás, 73, 81

N

nagy omega, *20*
nagy ordó, *19*, 242
nagy ordó nagy valószínűséggel, *51*
nagy teta, *20*, 243
nagy valószínűség, *50*
nagy valószínűségű nagy omega, 243
nagy valószínűségű nagy ordó, 242
nagy valószínűségű nagy teta, 243
négyzet, *39*, 119, *120*
N´ - - ´ ¨ , *147*
N´ - , *135*
N´ - ´ , *88*
négyzetes lépésszám, *21*
nem_ vezető, 201
nem ismétlődő úthalmaz, *176*
Nemzeti Periódika Adatbázis, 292
Net, 278
neurális hálók, 236
NPA, 292, *lásd* Nemzeti Periódika Adatbázis
nulla-egy elv, *100*

O, Ó

O - - , 217
O - - , 220
oszlopfolytonos indexelés, 135

Ö, Ő

összefésülés, 144, 189, 190
hiperkockán, 190
pillangó hálózaton, 190
összefüggő komponensek, 195
összehangolt támadás, 212
összetett rács, 40

P

palindróma, *160*

P´ - ´ -¨ ´ ¨ , 98

páratlan alhálózat, *183*

páratlan-páros összefésülés, 145

parciális permutációs csomagirányítás, 122

párhuzamos adatfeldolgozás, 15

párhuzamos algoritmus, 19, 242

párhuzamos algoritmus gyorsítása, 241

párhuzamos algoritmus hatékonysága, 241

párhuzamos algoritmus legrosszabb lépésszáma, 241–243

párhuzamos algoritmus legrosszabb üzenetszáma, 241

P´ - , *34*

P´ -´ , *35*

P´ - , 309

párhuzamos gép, *32*

párhuzamos hiperkocka, *164*

párhuzamos közvetlen hozzáférésű gép, 32

páros alhálózat, *183*

permutáció inverze, 58

permutációs hálózat, *42*

P ´- - , 193

pillangó hálózat, 42, 165, 192

piramis, *41*

polilogaritmikus lépésszám, *21*

polinomiális lépésszám, *21*

pontos bonyolultság, *26*

PPR, 122, 124, 129

PRAM, 31, 32, 122, 139, 144, 148, *lásd* párhuzamos közvetlen hozzáférésű gép

P - ´ , *180*

prefixszámítás, 73, 132, 133, 179
- bináris fán, 161
- hiperkockán, 179

Prim-algoritmus, 116

probléma mérete, 242

processzorok
- aszinkron, 16
- hibrid, 16
- részben aszinkron, 16
- szinkron, 16

processzorok indexelése, 135

processzor szintje, *165*

pszeudokód, 227

R

rács, 17, 119
  egyszerű, 40
  háromdimenziós, 40
  k-dimenziós, 39, 119
  kocka alakú, 119, 121, 151
  lánc alakú, 119
  négyzet alakú, 39, 119, 120
  összetett, 40
  tégla alakú, 121
  téglalap alakú, 39, 119, 120
RAM, *lásd* közvetlen hozzáférésű gép
RAN, *122*
rekurzió, *45*, *48*
rekurziótétel, 48
rekurzív algoritmus, *45*
relatív lépésszám, *24*
  lineáris, 24
  szublineáris, 24
  szuperlineáris, 24
relatív sebesség, 116
rendezés, 192
  biton sorozaté, 192
  csoportokba, 160
  hiperkockán, 192
  láncon, 160
  pillangó hálózaton, 192
  ritka, 137
  soronként, 161
  topologikus, 161
részben szinkronizált processzorok, 16
ritka leszámláló rendezés, *137*, 138, 183
R - , *138*
ritka rendezés, 132, 137
rossz algoritmus, *50*

S

síkgráf, *38*
síkhálózat, *38*
SIMD, *31*
SISD, *31*
sorfolytonos indexelés, 135
sorindex, *165*
sorméret, 178
soros adatfeldolgozás, 15
soros algoritmus, 19, 241
soros algoritmus legrosszabb lépésszáma, 241, 243
soros algoritmus szükséges lépésszáma, 242

soros hiperkocka, *164*
specifikáció, 15, *lásd* feladat megfogalmazása
Steelman Report, 232
súlyozott médián, *142*

SZ

szabad sorozat, *126*
számítási modell, 22
szinkronizált processzorok, 16
szublineáris lépésszám, *21*
szublineáris relatív lépésszám, *24*
szubpolinomiális lépésszám, *21*
szuffixszámítás, 159
szuperlineáris lépésszám, *21*
szuperlineáris relatív lépésszám, *24*
szuperlogaritmikus lépésszám, *21*
szuperpolinomiális lépésszám, *21*

T

tégla, *40*
téglalap, *39*, *120*
teljes keverő-cserélő hálózat, *199*
térgráf, *38*
térhálózat, *38*
topologikus rendezés, 161
torlódás, 170
tórusz, *40*, 160, 170, 221
tömb feje, 80
tömbrangsorolás, *79*
tranzitív lezárt, 152–154, 195

Ü, Ű

üzenetszórás, *132*, 179

V

valószínűségi feltétel, 216
váltási hely, *26*
V´ ´- ´ , 222
várakozási sor hossza, 123
véges ábécé, 243
V´ - , 81
V´ , 53
V´ - - ´ ´ , 178
V´ ´ - ´ ´ , 219
véletlenített algoritmus, 140, 216
véletlen választás, 122
vezető, 201
vezető processzor, 201

vezetőválasztás, *200*, 216, 221
gyűrűben, 201
négyzeten, 221
tóruszon, 221
vezetőválasztás hiperkockán, 221
vonalhiba,

X

XTANGO, 289